

**يوسف أحمد محمد ابراهيم**

**Assignment 6**  
**EXTRA**

# 1. ALSU design:

```
module ALSU(alsu_if alsuif);

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;

logic [5:0] sr_dataout;
logic mode;
assign mode = (opcode_reg == 3'h4) ? 0 : 1;

shift_reg sr(serial_in_reg, direction_reg, mode, alsuif.out, sr_dataout);

wire invalid_red_op, invalid_opcode, invalid;

//Invalid handling
assign invalid_red_op = (red_op_A_reg | red_op_B_reg) & (opcode_reg[1] | opcode_reg[2]);
assign invalid_opcode = opcode_reg[1] & opcode_reg[2];
assign invalid = invalid_red_op | invalid_opcode;

//Registering input signals
always @(posedge alsuif.clk or posedge alsuif.rst) begin
    if(alsuif.rst) begin
        cin_reg <= 0;
        red_op_B_reg <= 0;
        red_op_A_reg <= 0;
        bypass_B_reg <= 0;
        bypass_A_reg <= 0;
        direction_reg <= 0;
        serial_in_reg <= 0;
        opcode_reg <= 0;
        A_reg <= 0;
        B_reg <= 0;
    end else begin
        cin_reg <= alsuif.cin;
        red_op_B_reg <= alsuif.red_op_B;
        red_op_A_reg <= alsuif.red_op_A;
        bypass_B_reg <= alsuif.bypass_B;
        bypass_A_reg <= alsuif.bypass_A;
        direction_reg <= alsuif.direction;
        serial_in_reg <= alsuif.serial_in;
        opcode_reg <= alsuif.opcode;
        A_reg <= alsuif.A;
        B_reg <= alsuif.B;
    end
end
end
```



# 1. ALSU design:

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

//ALSU output processing
always @(posedge alsuif.clk or posedge alsuif.rst) begin
    if(alsuif.rst) begin
        alsuif.out <= 0;
    end
    else begin
        if (bypass_A_reg && bypass_B_reg)
            alsuif.out <= (alsuif.INPUT_PRIORITY == "A")? A_reg: B_reg;
        else if (bypass_A_reg)
            alsuif.out <= A_reg;
        else if (bypass_B_reg)
            alsuif.out <= B_reg;
        else if (invalid)
            alsuif.out <= 0;
        else begin
            case (opcode_reg)
                3'h0: begin
                    if (red_op_A_reg && red_op_B_reg)
                        alsuif.out <= (alsuif.INPUT_PRIORITY == "A")? |A_reg: |B_reg;
                    else if (red_op_A_reg)
                        alsuif.out <= |A_reg;
                    else if (red_op_B_reg)
                        alsuif.out <= |B_reg;
                    else
                        alsuif.out <= A_reg | B_reg;
                end
                3'h1: begin
                    if (red_op_A_reg && red_op_B_reg)
                        alsuif.out <= (alsuif.INPUT_PRIORITY == "A")? ^A_reg: ^B_reg;
                    else if (red_op_A_reg)
                        alsuif.out <= ^A_reg;
                    else if (red_op_B_reg)
                        alsuif.out <= ^B_reg;
                    else
                        alsuif.out <= A_reg ^ B_reg;
                end
                3'h2: alsuif.out <= (alsuif.FULL_ADDER)? (A_reg + B_reg + cin_reg) : A_reg + B_reg;
                3'h3: alsuif.out <= A_reg * B_reg;
                3'h4: alsuif.out <= sr_dataout;
                3'h5: alsuif.out <= sr_dataout;
                default : alsuif.out <= 0;
            endcase
        end
    end
end
endmodule
```

## 2. SHIFT\_REG design:

```
module shift_reg (serial_in, direction, mode, datain, dataout);
    input serial_in, direction, mode;
    input [5:0] datain;
    output reg [5:0] dataout;

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



## 2. ALSU Interface:

```
import shared_pkg::*;

interface alsu_if(input clk);

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

endinterface
```

## 2. SHIF\_REG Interface:

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



### 3. ALSU Top:

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

module top();

    bit clk;
    always #1 clk = ~clk;

    alsu_if alsuif(clk);
    sr_if srif();

    ALSU dut(alsuif);

    assign srif.serial_in = dut.serial_in_reg;
    assign srif.direction = dut.direction_reg;
    assign srif.mode = dut.mode;
    assign srif.datain = alsuif.out;
    assign srif.dataout = dut.sr_dataout;

    bind ALSU SVA SVA_inst(alsuif);

    initial begin
        uvm_config_db#(virtual alsu_if)::set(null, "uvm_test_top", "ALSU_VIF", alsuif);
        uvm_config_db#(virtual sr_if)::set(null, "uvm_test_top", "SR_VIF", srif);
        run_test("alsu_test");
    end
endmodule
```

### 3. Shared package:

```
1  package shared_pkg;
2      typedef enum {
3          OR,
4          XOR,
5          ADD,
6          MULT,
7          SHIFT,
8          ROTATE,
9          INVALID_6,
10         INVALID_7
11     } opcode_e;
12
13     parameter MAXPOS = 3'b011;
14     parameter ZERO = 3'b000;
15     parameter MAXNEG = 3'b100;
16 endpackage
```



## 4. SVA:

```
import shared_pkg::*;

module SVA (alsu_if alsuif);

    property p1;
        @(posedge alsuif.clk) disable iff(alsuif.rst || alsuif.bypass_A || alsuif.bypass_B)
            (alsuif.opcode == OR && alsuif.red_op_A)
            |-> ##2 (alsuif.out == (!$past(alsuif.A, 2)));
    endproperty

    property p2;
        @(posedge alsuif.clk) disable iff(alsuif.rst || alsuif.bypass_A || alsuif.bypass_B || alsuif.red_op_A)
            (alsuif.opcode == OR && alsuif.red_op_B)
            |-> ##2 (alsuif.out == (!$past(alsuif.B, 2)));
    endproperty

    property p3;
        @(posedge alsuif.clk) disable iff(alsuif.rst || alsuif.bypass_A || alsuif.bypass_B || alsuif.red_op_B || alsuif.red_op_A)
            (alsuif.opcode == OR)
            |-> ##2 (alsuif.out == ($past(alsuif.B, 2)) | ($past(alsuif.A, 2)));
    endproperty

    property p4;
        @(posedge alsuif.clk) disable iff(alsuif.rst || alsuif.bypass_A || alsuif.bypass_B || alsuif.red_op_B)
            (alsuif.opcode == XOR && alsuif.red_op_A)
            |-> ##2 (alsuif.out == (^$past(alsuif.A, 2)));
    endproperty

    property p5;
        @(posedge alsuif.clk) disable iff(alsuif.rst || alsuif.bypass_A || alsuif.bypass_B || alsuif.red_op_A)
            (alsuif.opcode == XOR && alsuif.red_op_B)
            |-> ##2 (alsuif.out == (^$past(alsuif.B, 2)));
    endproperty

    property p6;
        @(posedge alsuif.clk) disable iff(alsuif.rst || alsuif.bypass_A || alsuif.bypass_B || alsuif.red_op_B || alsuif.red_op_A)
            (alsuif.opcode == XOR)
            |-> ##2 (alsuif.out == ($past(alsuif.A, 2)) ^ ($past(alsuif.B, 2)));
    endproperty

    property p7;
        @(posedge alsuif.clk) disable iff(alsuif.rst || alsuif.bypass_A || alsuif.bypass_B || alsuif.red_op_B || alsuif.red_op_A)
            (alsuif.opcode == ADD)
            |-> ##2 (alsuif.out == $past(alsuif.B, 2) + $past(alsuif.A, 2) + $past($signed({0, alsuif.cin}), 2));
    endproperty

    property p8;
        @(posedge alsuif.clk) disable iff(alsuif.rst || alsuif.bypass_A || alsuif.bypass_B || alsuif.red_op_B || alsuif.red_op_A)
            (alsuif.opcode == MULT)
            |-> ##2 (alsuif.out == (($past(alsuif.A, 2)) * ($past(alsuif.B, 2))));
    endproperty

    property p9;
        @(posedge alsuif.clk) disable iff(alsuif.rst || alsuif.bypass_A || alsuif.bypass_B || alsuif.red_op_B || alsuif.red_op_A)
            ((alsuif.opcode == SHIFT) && alsuif.direction)
            |-> ##2 (alsuif.out == ({ $past(alsuif.out[4:0]), $past(alsuif.serial_in, 2) });
    endproperty

    property p10;
        @(posedge alsuif.clk) disable iff(alsuif.rst || alsuif.bypass_A || alsuif.bypass_B || alsuif.red_op_B || alsuif.red_op_A)
            ((alsuif.opcode == SHIFT) && !alsuif.direction)
            |-> ##2 (alsuif.out == ({ $past(alsuif.serial_in, 2), $past(alsuif.out[5:1]) });
    endproperty

endmodule
```

```
always_comb begin
    if(alsuif.rst)
        a_rst: assert final(alsuif.out == 0 && alsuif.leds == 0);
end
```



## 4. SVA:

```
property p10;
    @(posedge alsuif.clk) disable iff(alsuif.rst || alsuif.bypass_A || alsuif.bypass_B || alsuif.red_op_B || alsuif.red_op
    ((alsuif.opcode == SHIFT) && !alsuif.direction)
    |-> ##2 (alsuif.out == ({$past(alsuif.serial_in,2), $past(alsuif.out[5:1])});
endproperty

property p11;
    @(posedge alsuif.clk) disable iff(alsuif.rst || alsuif.bypass_A || alsuif.bypass_B || alsuif.red_op_B || alsuif.red_op
    ((alsuif.opcode == ROTATE) && alsuif.direction)
    |-> ##2 (alsuif.out == {$past(alsuif.out[4:0]), $past(alsuif.out[5])});
endproperty

property p12;
    @(posedge alsuif.clk) disable iff(alsuif.rst || alsuif.bypass_A || alsuif.bypass_B || alsuif.red_op_B || alsuif.red_op
    ((alsuif.opcode == ROTATE) && !alsuif.direction)
    |-> ##2 (alsuif.out == {$past(alsuif.out[0]), $past(alsuif.out[5:1])});
endproperty

property p13;
    @(posedge alsuif.clk) disable iff(alsuif.rst || alsuif.bypass_A || alsuif.bypass_B)
    ((alsuif.opcode != INVALID_6 && alsuif.opcode != INVALID_7 && alsuif.opcode != OR && alsuif.opcode != XOR)
    && (alsuif.red_op_B || alsuif.red_op_A))
    |-> ##2 (alsuif.out == 0);
endproperty

property p14;
    @(posedge alsuif.clk) disable iff(alsuif.rst || alsuif.bypass_A || alsuif.bypass_B)
    (alsuif.opcode == INVALID_6 || alsuif.opcode == INVALID_7)
    |-> ##2 (alsuif.out == 0);
endproperty

property p15;
    @(posedge alsuif.clk) disable iff(alsuif.rst)
    (alsuif.bypass_A) |-> ##2 (alsuif.out == $past(alsuif.A, 2));
endproperty

property p16;
    @(posedge alsuif.clk) disable iff(alsuif.rst || alsuif.bypass_A)
    (alsuif.bypass_B) |-> ##2 (alsuif.out == $past(alsuif.B, 2));
endproperty

a1: assert property (p1) else $display("ERROR1");
a2: assert property (p2) else $display("ERROR2");
a3: assert property (p3) else $display("ERROR3");
a4: assert property (p4) else $display("ERROR4");
a5: assert property (p5) else $display("ERROR5");
a6: assert property (p6) else $display("ERROR6");
a7: assert property (p7) else $display("ERROR7");
a8: assert property (p8) else $display("ERROR8");
a9: assert property (p9) else $display("ERROR9");
a10: assert property (p10) else $display("ERROR10");
a11: assert property (p11) else $display("ERROR11");
a12: assert property (p12) else $display("ERROR12");
a13: assert property (p13) else $display("ERROR13");
a14: assert property (p14) else $display("ERROR14");
a15: assert property (p15) else $display("ERROR15");
a16: assert property (p16) else $display("ERROR16");

c1: cover property (p1);
c2: cover property (p2);
c3: cover property (p3);
c4: cover property (p4);
c5: cover property (p5);
c6: cover property (p6);
c7: cover property (p7);
c8: cover property (p8);
c9: cover property (p9);
c10: cover property (p10);
c11: cover property (p11);
c12: cover property (p12);
c13: cover property (p13);
c14: cover property (p14);
c15: cover property (p15);
c16: cover property (p16);
endmodule
```



## 5. ALSU Configuration object:

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

    class alsu_config extends uvm_object;
        `uvm_object_utils(alsu_config);
        virtual alsu_if vif;
        uvm_active_passive_enum is_active;

        function new(string name = "alsu_config");
            super.new(name);
        endfunction
    endclass
endpackage
```

## 5. SHIFT REG Configuration object:

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

    class alsu_config extends uvm_object;
        `uvm_object_utils(alsu_config);
        virtual alsu_if vif;
        uvm_active_passive_enum is_active;

        function new(string name = "alsu_config");
            super.new(name);
        endfunction
    endclass
endpackage
```



## 6. ALSU Sequence item:

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

class alsu_item extends uvm_sequence_item;
    `uvm_object_utils(alsu_item);

    randc opcode_e      opcode_array[6];
    rand logic signed [2:0] A;
    rand logic signed [2:0] B;
    rand logic          cin;
    rand logic          serial_in;
    rand logic          red_op_A;
    rand logic          red_op_B;
    rand opcode_e       opcode;
    rand logic          bypass_A;
    rand logic          bypass_B;
    rand logic          rst;
    rand logic          direction;
    logic [15:0] leds;
    logic signed [5:0] out;

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

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

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

    constraint reset_c {
        rst dist {0 := 9, 1 := 1};
    }

    constraint invalid_c {
        opcode dist {[OR:ROTATE] := 5, [INVALID_6:INVALID_7] := 1};
    }

    constraint bypass_c {
        bypass_A dist {1 := 1, 0 := 7};
        bypass_B dist {1 := 1, 0 := 7};
    }
endclass
```



## 6. ALSU Sequence item:

```
constraint OR_XOR {
    if(opcode == OR || opcode == XOR) {
        if(red_op_A) { //priority for A so red_op_B here doesn't matter
            A dist {
                [3'b000:3'b111] := 1,
                3'b001 := 2,
                3'b010 := 2,
                3'b100 := 2
            };
            B == 3'b000;
        }

        else if(red_op_B){
            A == 3'b000;
            B dist {
                [3'b000:3'b111] := 1,
                3'b001 := 2,
                3'b010 := 2,
                3'b100 := 2
            };
        }
    }

    else {
        red_op_A dist {0 := 7, 1 := 3};
        red_op_B dist {0 := 7, 1 := 3};
    }
}

constraint add_mult_c {
    if(opcode == ADD || opcode == MULT) {
        A dist {[3'b001:3'b110] := 1, MAXPOS := 2, ZERO := 2, MAXNEG := 2};
        B dist {[3'b001:3'b110] := 1, MAXPOS := 2, ZERO := 2, MAXNEG := 2};
    }
}

constraint opcode_array_c {
    foreach (opcode_array[i])
        opcode_array[i] inside {SHIFT, ROTATE, ADD, MULT, OR, XOR};
}

endclass
endpackage
```



## 6. SR Sequence item:

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

class sr_seq_item extends uvm_sequence_item;
    `uvm_object_utils(sr_seq_item);

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

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

    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("reset = %0b, serial_in = %0b, direction = %0b, mode = %0b, datain = %0b",
            reset, serial_in, direction, mode, datain);
    endfunction
endclass
```



## 7. ALSU Sequences:

```
1  package sequence_pkg;
2      import shared_pkg::*;
3      import item_pkg::*;
4      import uvm_pkg::*;
5      `include "uvm_macros.svh"
6
7      class reset_sequence extends uvm_sequence #(alsu_item);
8          `uvm_object_utils(reset_sequence);
9
10         alsu_item reset_item;
11
12         function new(string name = "reset_sequence");
13             super.new(name);
14         endfunction
15
16         task body();
17             reset_item = alsu_item::type_id::create("reset_item");
18             start_item(reset_item);
19             reset_item.A = 0;
20             reset_item.B = 0;
21             reset_item.cin = 0;
22             reset_item.serial_in = 0;
23             reset_item.red_op_A = 0;
24             reset_item.red_op_B = 0;
25             reset_item.opcode = OR;
26             reset_item.bypass_A = 0;
27             reset_item.bypass_B = 0;
28             reset_item.rst = 1;
29             reset_item.direction = 0;
30             reset_item.leds = 0;
31             reset_item.out = 0;
32             finish_item(reset_item);
33         endtask
34
35     endclass
```

## 7. ALSU Sequences:

```
class main_sequence extends uvm_sequence #(alsu_item);
  `uvm_object_utils(main_sequence);

  alsu_item main_item;

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

  task body();

    main_item = alsu_item::type_id::create("main_item");
    main_item.opcode_array_c.constraint_mode(0);
    repeat(50000) begin
      start_item(main_item);
      assert(main_item.randomize());
      finish_item(main_item);
    end

    start_item(main_item);
    main_item.rst = 0;
    main_item.opcode = OR;
    finish_item(main_item);
    start_item(main_item);
    main_item.rst = 0;
    main_item.opcode = XOR;
    finish_item(main_item);

    start_item(main_item);
    main_item.rst = 0;
    main_item.opcode = ADD;
    finish_item(main_item);

    start_item(main_item);
    main_item.rst = 0;
    main_item.opcode = MULT;
    finish_item(main_item);

    start_item(main_item);
    main_item.rst = 0;
    main_item.opcode = SHIFT;
    finish_item(main_item);

    start_item(main_item);
    main_item.rst = 0;
    main_item.opcode = ROTATE;
    finish_item(main_item);
  end
endclass
```



## 7. ALSU Sequences:

```
main_item.constraint_mode(0);
main_item.opcode_array_c.constraint_mode(1);
repeat (10000) begin
    assert(main_item.randomize());

    foreach (main_item.opcode_array[j]) begin
        start_item(main_item);

        main_item.rst      = 0;
        main_item.bypass_A = 0;
        main_item.bypass_B = 0;
        main_item.red_op_A = 0;
        main_item.red_op_B = 0;

        main_item.opcode = main_item.opcode_array[j];

        finish_item(main_item);
    end
end
endtask
endclass
endpackage
```

## 8. ALSU Sequencer:

```
1 package sequencer_pkg;
2     import item_pkg::*;
3     import uvm_pkg::*;
4     `include "uvm_macros.svh"
5
6     class alsu_sequencer extends uvm_sequencer #(alsu_item);
7         `uvm_component_utils(alsu_sequencer);
8
9         function new(string name = "alsu_sequencer", uvm_component parent = null);
10             super.new(name, parent);
11         endfunction
12     endclass
13 endpackage
```

## 9. ALSU Driver:

```
1 package alsu_driver_pkg;
2     import uvm_pkg::*;
3     import item_pkg::*;
4     `include "uvm_macros.svh"
5
6     class alsu_driver extends uvm_driver #(alsu_item);
7         `uvm_component_utils(alsu_driver);
8         virtual alsu_if vif;
9         alsu_item driver_item;
10        function new(string name = "alsu_driver", uvm_component parent = null);
11            super.new(name, parent);
12        endfunction
13
14        task run_phase(uvm_phase phase);
15            super.run_phase(phase);
16            forever begin
17                driver_item = alsu_item::type_id::create("driver_item");
18                seq_item_port.get_next_item(driver_item);
19                vif.A = driver_item.A;
20                vif.B = driver_item.B;
21                vif.cin = driver_item.cin;
22                vif.serial_in = driver_item.serial_in;
23                vif.red_op_A = driver_item.red_op_A;
24                vif.red_op_B = driver_item.red_op_B;
25                vif.opcode = driver_item.opcode;
26                vif.bypass_A = driver_item.bypass_A;
27                vif.bypass_B = driver_item.bypass_B;
28                vif.direction = driver_item.direction;
29                vif.rst = driver_item.rst;
30                seq_item_port.item_done();
31                @(negedge vif.clk);
32                @(negedge vif.clk);
33                `uvm_info("driver_run_phase", driver_item.convert2string_stimulus(), UVM_HIGH)
34            end
35        endtask
36    endclass
37 endpackage
```



## 10. ALSU Monitor:

```
1  package alsu_monitor_pkg;
2      import item_pkg::*;
3      import uvm_pkg::*;
4      `include "uvm_macros.svh"
5
6      class alsu_monitor extends uvm_monitor;
7          `uvm_component_utils(alsu_monitor);
8          uvm_analysis_port #(alsu_item) mon_ap;
9          virtual alsu_if vif;
10         alsu_item item;
11
12         function new(string name = "alsu_monitor", 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             mon_ap = new("mon_ap", this);
19         endfunction
20
21         task run_phase(uvm_phase phase);
22             super.run_phase(phase);
23             forever begin
24                 item = alsu_item::type_id::create("item");
25                 @(negedge vif.clk);
26                 item.A = vif.A;
27                 item.B = vif.B;
28                 item.cin = vif.cin;
29                 item.serial_in = vif.serial_in;
30                 item.red_op_A = vif.red_op_A;
31                 item.red_op_B = vif.red_op_B;
32                 item.opcode = vif.opcode;
33                 item.bypass_A = vif.bypass_A;
34                 item.bypass_B = vif.bypass_B;
35                 item.rst = vif.rst;
36                 item.direction = vif.direction;
37                 item.leds = vif.leds;
38                 item.out = vif.out;
39                 mon_ap.write(item);
40                 `uvm_info("MONITOR", item.convert2string(), UVM_HIGH);
41             end
42         endtask
43     endclass
44 endpackage
```

# 10. SHIFT REG Monitor:

```
package sr_monitor_pkg;

import seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class sr_monitor extends uvm_monitor;
    `uvm_component_utils(sr_monitor);
    sr_seq_item monitor_item;
    virtual sr_if vif;
    uvm_analysis_port #(sr_seq_item) mon_ap;

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

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

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever begin
            monitor_item = sr_seq_item::type_id::create("monitor_item");
            #2;
            monitor_item.serial_in = vif.serial_in;
            monitor_item.direction = vif.direction;
            monitor_item.mode = vif.mode;
            monitor_item.datain = vif.datain;
            monitor_item.dataout = vif.dataout;
            mon_ap.write(monitor_item);
            `uvm_info("run_phase", monitor_item.convert2string(), UVM_HIGH);
        end
    endtask
endclass
endpackage
```



# 11. ALSU Agent:

```
package agent_pkg;
import alsu_driver_pkg::*;
import alsu_monitor_pkg::*;
import sequencer_pkg::*;
import alsu_config_pkg::*;
import item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"

class alsu_agent extends uvm_agent;
    `uvm_component_utils(alsu_agent);
    alsu_driver drv;
    alsu_monitor mon;
    alsu_sequencer sqr;
    alsu_config alsu_cfg;
    uvm_analysis_port #(alsu_item) agt_ap;

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

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        if(!uvm_config_db#(alsu_config)::get(this, "", "ALSU_CFG", alsu_cfg))
            `uvm_fatal("agent_build", "Unable to get configuration object!");

        if(alsu_cfg.is_active == UVM_ACTIVE) begin
            drv = alsu_driver::type_id::create("drv", this);
            sqr = alsu_sequencer::type_id::create("sqr", this);
        end
        mon = alsu_monitor::type_id::create("mon", this);
        agt_ap = new("agt_ap", this);
    endfunction

    function void connect_phase(uvm_phase phase);
        super.connect_phase(phase);
        mon.mon_ap.connect(agt_ap);

        if(alsu_cfg.is_active == UVM_ACTIVE) begin
            drv.seq_item_port.connect(sqr.seq_item_export);
            drv.vif = alsu_cfg.vif;
        end

        mon.vif = alsu_cfg.vif;
    endfunction
endclass
endpackage
```



# 11. SHIFT REG Agent:

```
package sr_agent_pkg;
import uvm_pkg::*;
import seq_item_pkg::*;
import sr_monitor_pkg::*;
import sr_driver_pkg::*;
import sr_config_pkg::*;
import sr_sequencer_pkg::*;
`include "uvm_macros.svh"

class sr_agent extends uvm_agent;
    `uvm_component_utils(sr_agent);
    sr_sequencer sqr;
    sr_driver drv;
    sr_monitor mon;
    sr_config sr_cfg;
    uvm_analysis_port #(sr_seq_item) agent_ap;

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

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        if(!uvm_config_db #(sr_config)::get(this, "", "SR_CFG", sr_cfg))
            `uvm_fatal("build_phase", "welcome to my error");
        if(sr_cfg.is_active == UVM_ACTIVE) begin
            sqr = sr_sequencer::type_id::create("sqr", this);
            drv = sr_driver::type_id::create("drv", this);
        end
        mon = sr_monitor::type_id::create("mon", this);
        agent_ap = new("agent_ap", this);
    endfunction

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

        if(sr_cfg.is_active == UVM_ACTIVE) begin
            drv.vif = sr_cfg.vif;
            drv.seq_item_port.connect(sqr.seq_item_export);
        end

        mon.vif = sr_cfg.vif;
        mon.mon_ap.connect(agent_ap);
    endfunction
endclass
endpackage
```



## 12. ALSU Scoreboard::

```
1 package scoreboard_pkg;
2 import shared_pkg::*;
3 import item_pkg::*;
4 import uvm_pkg::*;
5 `include "uvm_macros.svh"
6
7 class alsu_scoreboard extends uvm_scoreboard;
8     `uvm_component_utils(alsu_scoreboard);
9     uvm_analysis_export #(alsu_item) sb_export;
10    uvm_tlm_analysis_fifo #(alsu_item) sb_fifo;
11    alsu_item sb_item;
12    bit signed [2:0] A_reg, B_reg;
13    logic [2:0] opcode_reg;
14    bit red_op_A_reg, red_op_B_reg;
15    bit bypass_A_reg, bypass_B_reg;
16    bit direction_reg, serial_in_reg;
17    bit signed [1:0] cin_reg;
18    bit signed [5:0] expected_out; // holds previous out
19    logic [15:0] expected_leds;
20
21    int error_count, correct_count;
22
23    function new(string name = "alsu_scoreboard", 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        sb_export = new("sb_export", this);
30        sb_fifo = new("sb_fifo", this);
31    endfunction
32
33    function void connect_phase(uvm_phase phase);
34        super.connect_phase(phase);
35        sb_export.connect(sb_fifo.analysis_export);
36    endfunction
37
38    task run_phase(uvm_phase phase);
39        super.run_phase(phase);
40        forever begin
41            sb_fifo.get(sb_item);
42            ref_model(sb_item);
43            if (sb_item.out != expected_out || sb_item.leds != expected_leds) begin
44                `uvm_error("SCOREBOARD", $sformatf("Mismatch! out = %0h, leds = %0h, expected_out = %0h, expected_leds = %0h",
45                sb_item.out, sb_item.leds, expected_out, expected_leds));
46                error_count++;
47            end
48            else begin
49                `uvm_info("SCOREBOARD", $sformatf("Match! out = %0h, leds = %0h, expected_out = %0h, expected_leds = %0h",
50                sb_item.out, sb_item.leds, expected_out, expected_leds), UVM_LOW);
51                correct_count++;
52            end
53        end
54    endtask
```

## 12. ALSU Scoreboard::

```
56 task ref_model(alsu_item item);
57     bit invalid_red_op, invalid_opcode, invalid;
58
59     if (item.rst) begin
60         A_reg      = 0;
61         B_reg      = 0;
62         opcode_reg  = 0;
63         red_op_A_reg = 0;
64         red_op_B_reg = 0;
65         bypass_A_reg = 0;
66         bypass_B_reg = 0;
67         direction_reg = 0;
68         serial_in_reg = 0;
69         cin_reg      = 0;
70         expected_leds = 0;
71         expected_out  = 0;
72         return;
73     end
74
75     invalid_red_op = (red_op_A_reg | red_op_B_reg) & (opcode_reg[1] | opcode_reg[2]);
76     invalid_opcode = opcode_reg[1] & opcode_reg[2];
77     invalid        = invalid_red_op | invalid_opcode;
78
79     if (invalid)
80         expected_leds = ~expected_leds;
81     else
82         expected_leds = 0;
83
84
85     if (bypass_A_reg)
86         expected_out = A_reg;
87     else if (bypass_B_reg)
88         expected_out = B_reg;
89     else if (invalid)
90         expected_out = 0;
91     else begin
92         case (opcode_reg)
93             3'h0: begin
94
95                 if (red_op_A_reg)
96                     expected_out = {5'b0, |A_reg};
97                 else if (red_op_B_reg)
98                     expected_out = {5'b0, |B_reg};
99                 else
100                     expected_out = A_reg | B_reg;
101             end

```



## 12. ALSU Scoreboard:

```
102         3'h1: begin
103
104             if (red_op_A_reg)
105                 expected_out = {5'b0, ^A_reg};
106             else if (red_op_B_reg)
107                 expected_out = {5'b0, ^B_reg};
108             else
109                 expected_out = A_reg ^ B_reg;
110             end
111         3'h2: expected_out = (A_reg + B_reg + cin_reg);
112         3'h3: expected_out = A_reg * B_reg;
113         3'h4: begin
114             if (direction_reg)
115                 expected_out = {expected_out[4:0], serial_in_reg};
116             else
117                 expected_out = {serial_in_reg, expected_out[5:1]};
118             end
119         3'h5: begin
120             if (direction_reg)
121                 expected_out = {expected_out[4:0], expected_out[5]};
122             else
123                 expected_out = {expected_out[0], expected_out[5:1]};
124             end
125             default: expected_out = 0;
126         endcase
127     end
128
129     A_reg      = item.A;
130     B_reg      = item.B;
131     opcode_reg = item.opcode;
132     red_op_A_reg = item.red_op_A;
133     red_op_B_reg = item.red_op_B;
134     bypass_A_reg = item.bypass_A;
135     bypass_B_reg = item.bypass_B;
136     direction_reg = item.direction;
137     serial_in_reg = item.serial_in;
138     cin_reg      = item.cin;
139 endtask
140
141
142 function void report_phase(uvm_phase phase);
143     super.report_phase(phase);
144     `uvm_info("SCOREBOARD", $sformatf("Total Errors: %0d", error_count), UVM_LOW);
145     `uvm_info("SCOREBOARD", $sformatf("Total Correct: %0d", correct_count), UVM_LOW);
146 endfunction
147 endclass
148 endpackage
```

## 12. SHIFT REG Scoreboard:

```
package sr_scoreboard_pkg;

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

class sr_scoreboard extends uvm_scoreboard;
    `uvm_component_utils(sr_scoreboard);

    uvm_analysis_export #(sr_seq_item) sb_export;
    uvm_tlm_analysis_fifo #(sr_seq_item) sb_fifo;
    sr_seq_item sb_item;
    logic [5:0] expected_data;

    int error_coun = 0;
    int correct_count = 0;

    function new(string name, uvm_component parent);
        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(sb_item);
            ref_model(sb_item);
            if (sb_item.dataout !== expected_data) begin
                `uvm_error("SR_SCB", $sformatf("Mismatch! Expected: %0h, Got: %0h", expected_data, sb_item.dataout))
                error_coun++;
            end else begin
                `uvm_info("SR_SCB", $sformatf("Match! Expected: %0h, Got: %0h", expected_data, sb_item.dataout), UVM_HIGH)
                correct_count++;
            end
        end
    endtask

    task ref_model(sr_seq_item item);
        if (item.reset)
            expected_data = 0;
        else
            if (item.mode)
                if (item.direction)
                    expected_data = {item.datain[4:0], item.datain[5]};
                else
                    expected_data = {item.datain[0], item.datain[5:1]};
            else
                if (item.direction)
                    expected_data = {item.datain[4:0], item.serial_in};
                else
                    expected_data = {item.serial_in, item.datain[5:1]};
            endtask
    endtask

    function void report_phase(uvm_phase phase);
        super.report_phase(phase);
        `uvm_info("SR_SCB", $sformatf("total correct: %0d\ntotal errors: %0d", correct_count, error_coun), UVM_MEDIUM);
    endfunction
endclass
endpackage
```



## 13. ALSU Coverage collector:

```
1 package cov_pkg;
2     import shared_pkg::*;
3     import item_pkg::*;
4     import uvm_pkg::*;
5     `include "uvm_macros.svh"
6
7     class alsu_cov extends uvm_component;
8         `uvm_component_utils(alsu_cov);
9         uvm_analysis_export #(alsu_item) cov_export;
10        uvm_tlm_analysis_fifo #(alsu_item) cov_fifo;
11        alsu_item cov_item;
12
13    covergroup cvr_gp;
14
15        A_cp : coverpoint cov_item.A {
16            option.comment = "If only the red_op_A is high";
17            bins A_data_0      = {0};
18            bins A_data_max    = {MAXPOS};
19            bins A_data_min    = {MAXNEG};
20            bins A_data_default = default;
21            bins A_data_walkingones[] = {3'b001, 3'b010, 3'b100}
22            | iff (cov_item.red_op_A);
23
24        }
25
26        B_cp : coverpoint cov_item.B {
27            option.comment = "If only red_op_B is high and red_op_A is low";
28            bins B_data_0      = {0};
29            bins B_data_max    = {MAXPOS};
30            bins B_data_min    = {MAXNEG};
31            bins B_data_default = default;
32            bins B_data_walkingones[] = {3'b001, 3'b010, 3'b100}
33            | iff (cov_item.red_op_B && !cov_item.red_op_A);
34
35        }
36
37        opcode_transition : coverpoint cov_item.opcode {
38            bins Bins_trans = (OR [*2] => XOR[*2] => ADD[*2] => MULT[*2] => SHIFT[*2] => ROTATE[*2]);
39
40        }
41
42        ALU_cp : coverpoint cov_item.opcode {
43            bins Bins_shift[] = {SHIFT, ROTATE};
44            bins Bins_arith[] = {ADD, MULT};
45            bins Bins_bitwise[] = {OR, XOR};
46            illegal_bins Bins_invalid = {6, 7};
47
48        }
49
50        op_arth : coverpoint cov_item.opcode {
51            option.weight = 0;
52            bins ADD_b = {ADD};
53            bins MULT_b = {MULT};
54            bins shift = {SHIFT};
55
56        }
57    endclass
```



## 13. ALSU Coverage collector:

```
52     c_B : coverpoint cov_item.B {
53         option.weight = 0;
54         bins B_0      = {0};
55         bins B_max    = {MAXPOS};
56         bins B_min    = {MAXNEG};
57         bins walkingones1 = {3'b001};
58         bins walkingones2 = {3'b010};
59         bins walkingones3 = {3'b100};
60     }
61     c_A : coverpoint cov_item.A {
62         option.weight = 0;
63         bins A_0      = {0};
64         bins A_max    = {MAXPOS};
65         bins A_min    = {MAXNEG};
66         bins walkingones1 = {3'b001};
67         bins walkingones2 = {3'b010};
68         bins walkingones3 = {3'b100};
69     }
70
71     red_A: coverpoint cov_item.red_op_A;
72     red_B: coverpoint cov_item.red_op_B;
73     op: coverpoint cov_item.opcode;
74
75
76     c1: cross op_arth, c_B, c_A{
77         ignore_bins b1 = binsof(op_arth) intersect {SHIFT};
78         ignore_bins b2 = binsof(c_A.walkingones1);
79         ignore_bins b3 = binsof(c_A.walkingones2);
80         ignore_bins b4 = binsof(c_A.walkingones3);
81         ignore_bins b5 = binsof(c_B.walkingones1);
82         ignore_bins b6 = binsof(c_B.walkingones2);
83         ignore_bins b7 = binsof(c_B.walkingones3);
84     }
85
86     c2: cross op_arth, cov_item.cin {
87         ignore_bins b1 = binsof(op_arth) intersect {MULT, SHIFT};
88     }
89
90     c3: cross op_arth, cov_item.direction {
91         ignore_bins b1 = binsof(op_arth) intersect {MULT, SHIFT};
92     }
93
94
95     c4: cross op_arth, cov_item.serial_in {
96         ignore_bins b1 = binsof(op_arth) intersect {MULT, SHIFT};
97     }
```



## 13. ALSU Coverage collector:

```

99   c5: cross ALU_cp, red_A, c_A, c_B{
100   option.cross_auto_bin_max = 0;
101   bins b1 = binsof(ALU_cp.Bins_bitwise) intersect {OR} &&
102           binsof(red_A) intersect {1} &&
103           binsof(c_B) intersect {0} &&
104           binsof(c_A.walkingones1);
105   bins b2 = binsof(ALU_cp.Bins_bitwise) intersect {OR} &&
106           binsof(red_A) intersect {1} &&
107           binsof(c_B) intersect {0} &&
108           binsof(c_A.walkingones2);
109   bins b3 = binsof(ALU_cp.Bins_bitwise) intersect {OR} &&
110           binsof(red_A) intersect {1} &&
111           binsof(c_B) intersect {0} &&
112           binsof(c_A.walkingones3);
113   bins b4 = binsof(ALU_cp.Bins_bitwise) intersect {XOR} &&
114           binsof(red_A) intersect {1} &&
115           binsof(c_B) intersect {0} &&
116           binsof(c_A.walkingones1);
117   bins b5 = binsof(ALU_cp.Bins_bitwise) intersect {XOR} &&
118           binsof(red_A) intersect {1} &&
119           binsof(c_B) intersect {0} &&
120           binsof(c_A.walkingones2);
121   bins b6 = binsof(ALU_cp.Bins_bitwise) intersect {XOR} &&
122           binsof(red_A) intersect {1} &&
123           binsof(c_B) intersect {0} &&
124           binsof(c_A.walkingones3);
125   }
126
127
128   c6: cross ALU_cp, red_B, c_A, c_B{
129   option.cross_auto_bin_max = 0;
130
131   bins b1 = binsof(ALU_cp.Bins_bitwise) intersect {OR} &&
132           binsof(red_B) intersect {1} &&
133           binsof(c_A) intersect {0} &&
134           binsof(c_B.walkingones1);
135
136   bins b2 = binsof(ALU_cp.Bins_bitwise) intersect {OR} &&
137           binsof(red_B) intersect {1} &&
138           binsof(c_A) intersect {0} &&
139           binsof(c_B.walkingones2);
140
141   bins b3 = binsof(ALU_cp.Bins_bitwise) intersect {OR} &&
142           binsof(red_B) intersect {1} &&
143           binsof(c_A) intersect {0} &&
144           binsof(c_B.walkingones3);
145
146   bins b4 = binsof(ALU_cp.Bins_bitwise) intersect {XOR} &&
147           binsof(red_B) intersect {1} &&
148           binsof(c_A) intersect {0} &&
149           binsof(c_B.walkingones1);

```



## 13. ALSU Coverage collector:

```
151         bins b5 = binsof(ALU_cp.Bins_bitwise) intersect {XOR} &&
152         | binsof(red_B) intersect {1} &&
153         | binsof(c_A) intersect {0} &&
154         | binsof(c_B.walkingones2);
155
156         bins b6 = binsof(ALU_cp.Bins_bitwise) intersect {XOR} &&
157         | binsof(red_B) intersect {1} &&
158         | binsof(c_A) intersect {0} &&
159         | binsof(c_B.walkingones3);
160     }
161
162
163     c7: cross red_A, red_B, op {
164         ignore_bins b1 = binsof(red_A) intersect {0};
165         ignore_bins b2 = binsof(red_B) intersect {0};
166         ignore_bins b3 = binsof(op) intersect {OR, XOR};
167     }
168 }
169 endgroup
170
171
172     function new(string name = "alsu_cov", uvm_component parent = null);
173         super.new(name, parent);
174         cvr_gp = new;
175     endfunction
176
177     function void build_phase(uvm_phase phase);
178         super.build_phase(phase);
179         cov_export = new("cov_export", this);
180         cov_fifo = new("cov_fifo", this);
181     endfunction
182
183     function void connect_phase(uvm_phase phase);
184         super.connect_phase(phase);
185         cov_export.connect(cov_fifo.analysis_export);
186     endfunction
187
188     task run_phase(uvm_phase phase);
189         super.run_phase(phase);
190         forever begin
191             cov_fifo.get(cov_item);
192             `uvm_info("COVERAGE", $sformatf("OPCODE: %S", cov_item.opcode), UVM_LOW);
193             cvr_gp.sample();
194         end
195     endtask
196 endclass
197 endpackage
```



## 13. SHIFT REG Coverage collector:

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

class sr_cov_collector extends uvm_component;
    `uvm_component_utils(sr_cov_collector);
    uvm_analysis_export #(sr_seq_item) cov_export;
    uvm_tlm_analysis_fifo #(sr_seq_item) cov_fifo;
    sr_seq_item cov_item;

    covergroup sr_cg;
        direction_cp : coverpoint cov_item.direction;

        mode_cp : coverpoint cov_item.mode;

        datain_cp : coverpoint cov_item.datain;

        serial_in_cp : coverpoint cov_item.serial_in;

        dataout_cp : coverpoint cov_item.dataout;
    endgroup

    function new(string name, uvm_component parent);
        super.new(name, parent);
        sr_cg = 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(cov_item);
            sr_cg.sample();
        end
    endtask

endclass
endpackage
```



## 14. ALSU Environment:

```
package alsu_env_pkg;
    import agent_pkg::*;
    import scoreboard_pkg::*;
    import cov_pkg::*;
    import alsu_driver_pkg::*;
    import uvm_pkg::*;
    `include "uvm_macros.svh"

    class alsu_env extends uvm_env;
        `uvm_component_utils(alsu_env);
        alsu_agent agent;
        alsu_scoreboard sb;
        alsu_cov cov;
        function new(string name = "alsu_env", uvm_component parent = null);
            super.new(name, parent);
        endfunction

        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            agent = alsu_agent::type_id::create("agent", this);
            sb = alsu_scoreboard::type_id::create("sb", this);
            cov = alsu_cov::type_id::create("cov", this);
        endfunction

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



## 14. SHIFT REG Environment:

```
package sr_env_pkg;
  import sr_agent_pkg::*;
  import sr_scoreboard_pkg::*;
  import sr_cov_collector_pkg::*;
  import uvm_pkg::*;
  `include "uvm_macros.svh"

  class sr_env extends uvm_env;
    `uvm_component_utils(sr_env);
    sr_agent agent;
    sr_scoreboard sb;
    sr_cov_collector cov;

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

    function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      agent = sr_agent::type_id::create("agent", this);
      sb = sr_scoreboard::type_id::create("sb", this);
      cov = sr_cov_collector::type_id::create("cov", this);
    endfunction

    function void connect_phase(uvm_phase phase);
      super.connect_phase(phase);
      agent.agent_ap.connect(sb.sb_export);
      agent.agent_ap.connect(cov.cov_export);
    endfunction
  endclass
endpackage
```



# 15. ALSU Test:

```
package alsu_test_pkg;
import alsu_env_pkg::*;
import sr_env_pkg::*;
import sr_config_pkg::*;
import sequence_pkg::*;
import alsu_config_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"

class alsu_test extends uvm_test;
    `uvm_component_utils(alsu_test);
    alsu_env alsuEnv;
    sr_env srEnv;
    alsu_config alsu_cfg;
    sr_config sr_cfg;
    reset_sequence reset_seq;
    main_sequence main_seq;
    function new(string name = "alsu_test", uvm_component parent = null);
        super.new(name, parent);
    endfunction

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

        alsuEnv = alsu_env::type_id::create("alsuEnv", this);
        srEnv = sr_env::type_id::create("srEnv", this);
        alsu_cfg = alsu_config::type_id::create("alsu_cfg", this);
        sr_cfg = sr_config::type_id::create("sr_cfg", this);
        reset_seq = reset_sequence::type_id::create("reset_seq");
        main_seq = main_sequence::type_id::create("main_seq");

        if(!uvm_config_db#(virtual alsu_if)::get(this, "", "ALSU_VIF", alsu_cfg.vif))
            `uvm_fatal("ALSU_TEST", "Virtual interface not exist");

        if(!uvm_config_db#(virtual sr_if)::get(this, "", "SR_VIF", sr_cfg.vif))
            `uvm_fatal("ALSU_TEST", "Virtual interface not exist");

        uvm_config_db#(alsu_config)::set(this, "", "ALSU_CFG", alsu_cfg);
        uvm_config_db#(sr_config)::set(this, "", "SR_CFG", sr_cfg);

        alsu_cfg.is_active = UVM_ACTIVE;
        sr_cfg.is_active = UVM_PASSIVE;
    endfunction

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        phase.raise_objection(this);
        `uvm_info("ALSU_TEST", "Starting reset test", UVM_LOW);
        reset_seq.start(alsuEnv.agent.sqr);
        `uvm_info("ALSU_TEST", "Starting main test", UVM_LOW);
        main_seq.start(alsuEnv.agent.sqr);
        #10ns;
        `uvm_info("ALSU_TEST", "Test completed", UVM_LOW);
        phase.drop_objection(this);
    endtask
endclass
endpackage
```



## 16. Src file:

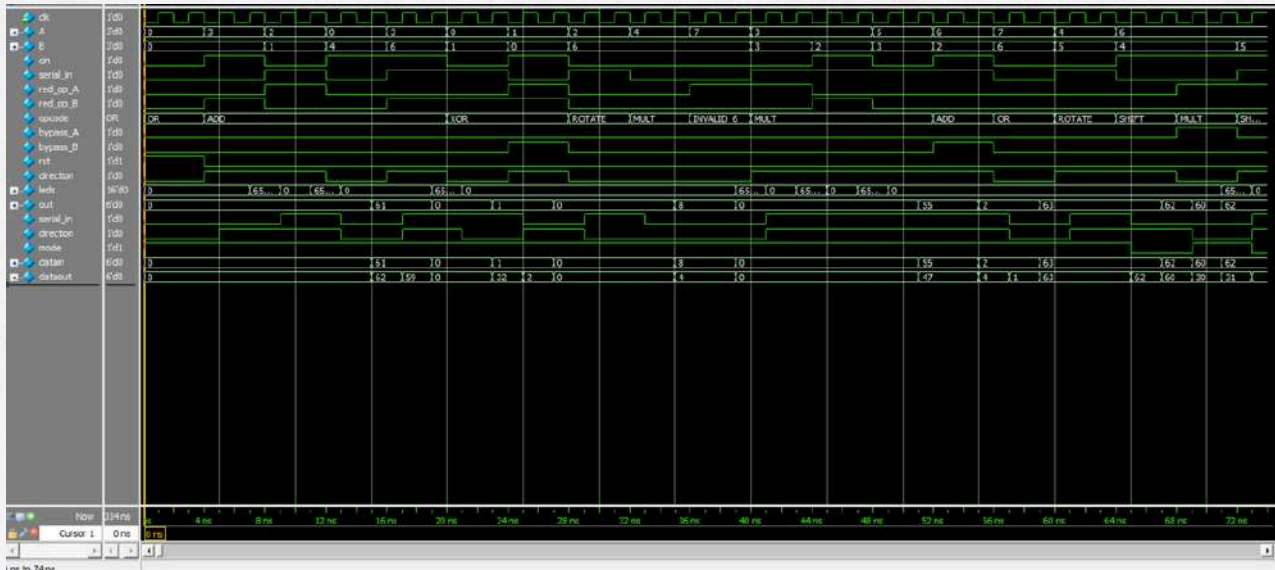
```
1  # ----- RTL + Interfaces -----
2  sr_if.sv
3  sr.v
4  alsu_if.sv
5  ALSU.sv
6  alsu_assertions.sv
7
8  # ----- SR UVM Environment -----
9  sr_sequence_item.sv
10 sr_seq.sv
11 sr_config.sv
12 sr_monitor.sv
13 sr_driver.sv
14 sr_sequencer.sv
15 sr_agent.sv
16 sr_scoreboard.sv
17 sr_cov_collector.sv
18 sr_env.sv
19 sr_test.sv
20 sr_top.sv
21
22 # ----- ALSU UVM Environment -----
23 alsu_pkg.sv
24 alsu_config.sv
25 alsu_item.sv
26 alsu_sequence.sv
27 alsu_sequencer.sv
28 alsu_driver.sv
29 alsu_monitor.sv
30 alsu_agent.sv
31 alsu_scoreboard.sv
32 alsu_cov.sv
33 alsu_env.sv
34 alsu_test.sv
35 alsu_top.sv
36
```

## 17. Do:

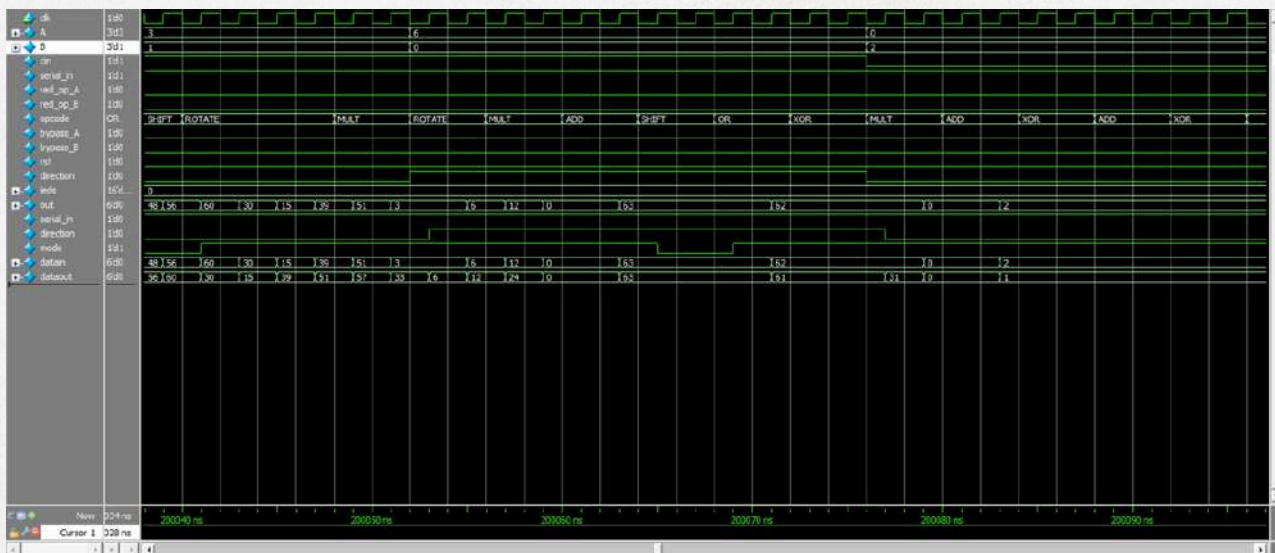
```
vlib work
vlog -cover bcestf -f alsu_src_files.list
vsim -coverage -voptargs=+acc work.top -classdebug -uvmcontrol=all
coverage save cov.ucdb -onexit
coverage exclude -src ALSU.sv -line 103 -code s
coverage exclude -src ALSU.sv -line 103 -code b
coverage exclude -du ALSU -togglenode {cin_reg[1]}
add wave -position insertpoint sim:/top/alsuif/*
add wave -position insertpoint sim:/top/srif/*
run -all
```

## 18. Questasim:

- Reset test and First loop:



- Second loop:





# 18. Questasim:

```
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :440044
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [ALU_TEST]      3
# [COVERAGE] 220017
# [Questa UVM]    2
# [RNTST]         1
# [SCOREBOARD] 220019
# [SR_SCB]        1
# [TEST_DONE]     1
# ** Note: $finish      : C:/questasim64_2021.1/win64/./verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#   Time: 440034 ns  Iteration: 54  Instance: /top
# 1
# Break in Task uvm_pkg/uvm_root::run_test at C:/questasim64_2021.1/win64/./verilog_src/uvm-1.1d/src/base/uvm_root.svh line 430
```



## 19. Code coverage:

- ALSU Statement coverage:

```
Statement Coverage:
  Enabled Coverage          Bins    Hits    Misses  Coverage
  -----
  Statements                47      47        0   100.00%

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

Statement Coverage for instance /top/dut --
```

- ALSU Branch coverage:

```
=====
=== Instance: /top/dut
=== Design Unit: work.ALSU
=====
✓ Branch Coverage:
  Enabled Coverage          Bins    Hits    Misses  Coverage
  -----
  Branches                 29      29        0   100.00%
```

- ALSU Toggle coverage:

```
Toggle Coverage:
  Enabled Coverage          Bins    Hits    Misses  Coverage
  -----
  Toggles                  52      52        0   100.00%

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



## 19. Code coverage:

- SR Statement coverage:

```
=====
=== Instance: /top/dut/sr
=== Design Unit: work.shift_reg
=====
Branch Coverage:
  Enabled Coverage          Bins    Hits    Misses  Coverage
  -----
  Branches                  5      5      0    100.00%
```

- SR Branch coverage:

```
Statement Coverage:
  Enabled Coverage          Bins    Hits    Misses  Coverage
  -----
  Statements                 5      5      0    100.00%

=====Statement Details=====
```

- SR Toggle coverage:

```
Toggle Coverage:
  Enabled Coverage          Bins    Hits    Misses  Coverage
  -----
  Toggles                   30     30      0    100.00%

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



## 20. SU Functional coverage:

Covergroup Coverage:

Covergroups	1	na	na	100.00%
Coverpoints/Crosses	20	na	na	na
Covergroup Bins	72	72	0	100.00%

Covergroup

Metric

Goal

Bins

Status

TYPE /cov\_pkg/alsu\_cov/cvr\_gp

100.00%

100

-

Covered

## 20. SR Functional coverage:

```
=====
=== Instance: /sr_cov_collector_pkg
=== Design Unit: work.sr_cov_collector_pkg
=====
```

Covergroup Coverage:

Covergroups	1	na	na	100.00%
Coverpoints/Crosses	5	na	na	na
Covergroup Bins	134	134	0	100.00%



# 21. Assertions:

## ASSERTION RESULTS:

Name	File(Line)	Failure Count	Pass Count
-----			
/top/dut/SVA_inst/a_rst			
	alsu_assertions.sv(7)	0	1
/top/dut/SVA_inst/a1	alsu_assertions.sv(126)	0	1
/top/dut/SVA_inst/a2	alsu_assertions.sv(127)	0	1
/top/dut/SVA_inst/a3	alsu_assertions.sv(128)	0	1
/top/dut/SVA_inst/a4	alsu_assertions.sv(129)	0	1
/top/dut/SVA_inst/a5	alsu_assertions.sv(130)	0	1
/top/dut/SVA_inst/a6	alsu_assertions.sv(131)	0	1
/top/dut/SVA_inst/a7	alsu_assertions.sv(132)	0	1
/top/dut/SVA_inst/a8	alsu_assertions.sv(133)	0	1
/top/dut/SVA_inst/a9	alsu_assertions.sv(134)	0	1
/top/dut/SVA_inst/a10			
	alsu_assertions.sv(135)	0	1
/top/dut/SVA_inst/a11			
	alsu_assertions.sv(136)	0	1
/top/dut/SVA_inst/a12			
	alsu_assertions.sv(137)	0	1
/top/dut/SVA_inst/a13			
	alsu_assertions.sv(138)	0	1
/top/dut/SVA_inst/a14			
	alsu_assertions.sv(139)	0	1
/top/dut/SVA_inst/a15			
	alsu_assertions.sv(140)	0	1
/top/dut/SVA_inst/a16			
	alsu_assertions.sv(141)	0	1
/top/dut/SVA_inst/a17			
	alsu_assertions.sv(142)	0	1
/top/dut/SVA_inst/a18			
	alsu_assertions.sv(143)	0	1
/top/dut/SVA_inst/a19			
	alsu_assertions.sv(144)	0	1
/sequence_pkg/main_sequence/body/#ublk#50851543#51/immed__53			
	alsu_sequence.sv(53)	0	1
/sequence_pkg/main_sequence/body/#ublk#50851543#88/immed__89			
	alsu_sequence.sv(89)	0	1



## 22. Assertions coverage:

### DIRECTIVE COVERAGE:

Name	Design Unit	Design UnitType	Lang	File(Line)	Hits	Status
/top/dut/SVA_inst/c1	SVA	Verilog	SVA	alsu_assertions.sv(147)	176	Covered
/top/dut/SVA_inst/c2	SVA	Verilog	SVA	alsu_assertions.sv(148)	58	Covered
/top/dut/SVA_inst/c3	SVA	Verilog	SVA	alsu_assertions.sv(149)	1235	Covered
/top/dut/SVA_inst/c4	SVA	Verilog	SVA	alsu_assertions.sv(150)	60	Covered
/top/dut/SVA_inst/c5	SVA	Verilog	SVA	alsu_assertions.sv(151)	42	Covered
/top/dut/SVA_inst/c6	SVA	Verilog	SVA	alsu_assertions.sv(152)	1234	Covered
/top/dut/SVA_inst/c7	SVA	Verilog	SVA	alsu_assertions.sv(153)	1212	Covered
/top/dut/SVA_inst/c8	SVA	Verilog	SVA	alsu_assertions.sv(154)	1182	Covered
/top/dut/SVA_inst/c9	SVA	Verilog	SVA	alsu_assertions.sv(155)	596	Covered
/top/dut/SVA_inst/c10	SVA	Verilog	SVA	alsu_assertions.sv(156)	574	Covered
/top/dut/SVA_inst/c11	SVA	Verilog	SVA	alsu_assertions.sv(157)	608	Covered
/top/dut/SVA_inst/c12	SVA	Verilog	SVA	alsu_assertions.sv(158)	572	Covered
/top/dut/SVA_inst/c13	SVA	Verilog	SVA	alsu_assertions.sv(159)	1482	Covered
/top/dut/SVA_inst/c14	SVA	Verilog	SVA	alsu_assertions.sv(160)	304	Covered
/top/dut/SVA_inst/c15	SVA	Verilog	SVA	alsu_assertions.sv(161)	946	Covered
/top/dut/SVA_inst/c16	SVA	Verilog	SVA	alsu_assertions.sv(162)	830	Covered
/top/dut/SVA_inst/c17	SVA	Verilog	SVA	alsu_assertions.sv(163)	2538	Covered
/top/dut/SVA_inst/c18	SVA	Verilog	SVA	alsu_assertions.sv(164)	526	Covered
/top/dut/SVA_inst/c19	SVA	Verilog	SVA	alsu_assertions.sv(165)	10939	Covered

TOTAL DIRECTIVE COVERAGE: 100.00% COVERS: 19