

UVM

Assignment6_extra

By: Mohamed Sayed Mohamed Soliman

#part 1

1- ALSU CODE Design

```
3 module ALSU(A, B, cin, serial_in, red_op_A, red_op_B, opcode, bypass_A, bypass_B, clk, rst, direction, leds, out);
4 parameter INPUT_PRIORITY = "A";
5 parameter FULL_ADDER = "ON";
6 input clk, rst, cin, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
7 input [2:0] opcode;
8 input signed [2:0] A, B; // first bug [ ]> we must put it signed
9 output reg [15:0] leds;
10 output reg signed [5:0] out; // second bug [ ]> we must put it signed
11 reg cin_reg, red_op_A_reg, red_op_B_reg, bypass_A_reg, bypass_B_reg, direction_reg, serial_in_reg;
12 reg [2:0] opcode_reg;
13 reg signed [2:0] A_reg, B_reg;
14 wire invalid_red_op, invalid_opcode, invalid;
15 //Invalid handling
16 assign invalid_red_op = (red_op_A_reg | red_op_B_reg) & (opcode_reg[1] | opcode_reg[2]);
17 assign invalid_opcode = opcode_reg[1] & opcode_reg[2];
18 assign invalid = invalid_red_op | invalid_opcode;
19 //Registering input signals
20 always @(posedge clk or posedge rst) begin
21     if(rst) begin
22         cin_reg <= 0;
23         red_op_B_reg <= 0;
24         red_op_A_reg <= 0;
25         bypass_B_reg <= 0;
26         bypass_A_reg <= 0;
27         direction_reg <= 0;
28         serial_in_reg <= 0;
29         opcode_reg <= 0;
30         A_reg <= 0;
31         B_reg <= 0;
32     end else begin
33         cin_reg <= cin;
34         red_op_B_reg <= red_op_B;
35         red_op_A_reg <= red_op_A;
36         bypass_B_reg <= bypass_B;
37         bypass_A_reg <= bypass_A;
38         direction_reg <= direction;
39         serial_in_reg <= serial_in;
40         opcode_reg <= opcode;
41         A_reg <= A;
42         B_reg <= B;
43     end
44 end
45 //leds output blinking
46 always @(posedge clk or posedge rst) begin
47     if(rst) begin
48         leds <= 0;
49     end else begin
50         if (invalid)
51             leds <= ~leds;
52         else
53             leds <= 0;
54     end
55 end
56 //ALSU output processing
```

```

57 always @(posedge clk or posedge rst) begin
58     if(rst) begin
59         out <= 0;
60     end
61     else begin
62         if (invalid)
63             out <= 0;
64         else if (bypass_A_reg && bypass_B_reg)
65             out <= (INPUT_PRIORITY == "A")? A_reg: B_reg;
66         else if (bypass_A_reg)
67             out <= A_reg;
68         else if (bypass_B_reg)
69             out <= B_reg;
70         else begin
71             case (opcode_reg) // third bug is to used the opcode_reg not the opcode
72                 3'h0: begin
73                     if (red_op_A_reg && red_op_B_reg)
74                         out <= (INPUT_PRIORITY == "A")? |A_reg: |B_reg; // third bug is to replace AND with OR
75                     else if (red_op_A_reg)
76                         out <= |A_reg;
77                     else if (red_op_B_reg)
78                         out <= |B_reg;
79                     else
80                         out <= A_reg | B_reg;
81                 end
82                 3'h1: begin
83                     if (red_op_A_reg && red_op_B_reg)
84                         out <= (INPUT_PRIORITY == "A")? ^A_reg: ^B_reg; // fourth bug is to replace OR with XOR
85                     else if (red_op_A_reg)
86                         out <= ^A_reg;
87                     else if (red_op_B_reg)
88                         out <= ^B_reg;
89                     else
90                         out <= A_reg ^ B_reg;
91                 end
92                 3'h2: begin
93                     if(ALSUif.FULL_ADDER == "ON") // fifth bug to add Cin operation in case of full adde
94                         ALSUif.out <= A_reg + B_reg + cin_reg ;
95                     else
96                         ALSUif.out <= A_reg + B_reg ;
97                     end
98                 3'h3: ALSUif.out <= A_reg * B_reg;
99                 3'h4: begin
100                     ALSUif.out <= ALSUif.out_shift_reg ;
101                 end
102                 3'h5: begin
103                     ALSUif.out <= ALSUif.out_shift_reg ;
104                 end
105             default : ALSUif.out <= 0 ;
106         endcase
107     end
108 end
109 end
110
111 endmodule

```

2- Shift code design

```
8  module shift_reg (shift_reg_if.DUT_shift shift_if );
9
10 always @(*) begin
11     if (shift_if.mode) // rotate
12         if (shift_if.direction) // left
13             shift_if.dataout <= {shift_if.datain[4:0], shift_if.datain[5]};
14         else
15             shift_if.dataout <= {shift_if.datain[0], shift_if.datain[5:1]};
16     else // shift
17         if (shift_if.direction) // left
18             shift_if.dataout <= {shift_if.datain[4:0], shift_if.serial_in};
19         else
20             shift_if.dataout <= {shift_if.serial_in, shift_if.datain[5:1]};
21 end
22 endmodule
```

3- ALSU Interface

```
1  interface ALSU_if (clk);
2      input clk ;
3      logic rst, cin, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
4      logic [2:0] opcode;
5      logic signed [2:0] A, B;
6      logic [15:0] leds;
7      logic signed [5:0] out;
8
9      modport DUT (input clk , rst, cin, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in , opcode , A , B , output leds , out ) ;
10
11  endinterface
```

4- Shift Interface

```
1  interface shift_reg_if ();
2      logic serial_in, direction, mode;
3      logic signed [5:0] datain, dataout;
4
5
6      modport DUT_shift (input direction, serial_in , mode , datain , output dataout ) ;
7  endinterface : shift_reg_if
```

5- top module

```
1  import ALSU_shift_test_pkg::*;
2  import uvm_pkg::* ;
3  `include "uvm_macros.svh"
4
5  module ALSU_shift_top();
6  bit clk ;
7
8  initial
9      begin
10         clk = 0 ;
11         forever
12             #1 clk = ~clk ;
13         end
14     ALSU_if ALSUif (clk) ;
15     shift_reg_if shift_if() ;
16     ALSU dut_AL (ALSUif) ;
17     shift_reg DUT_SH ( shift_if ) ;
18
19     assign shift_if.serial_in = ALSUif.serial_in ;
20     assign shift_if.direction = ALSUif.direction ;
21     assign shift_if.datain = ALSUif.out ;
22     assign shift_if.mode = ALSUif.opcode[0] ;
23     assign ALSUif.out_shift_reg = shift_if.dataout ;
24     bind ALSU SVA sva(ALSUif) ;
25     initial
26         begin
27             uvm_config_db#(virtual ALSU_if)::set(null , "uvm_test_top" , "ALSUif" , ALSUif );
28             uvm_config_db#(virtual shift_reg_if)::set(null , "uvm_test_top" , "SHIFT_IF" , shift_if );
29             run_test("test");
30         end
31
32
33 endmodule
```

6- alsu_test

```
1 package ALSU_shift_test_pkg ;
2 import ALSU_env_pkg::*;
3 import shift_reg_env_pkg::*;
4 import ALSU_config_pkg::*;
5 import shift_reg_config_pkg::*;
6 import ALSU_seq_reset_pkg::*;
7 import ALSU_seq_main_pkg::*;
8 import uvm_pkg::* ;
9 `include "uvm_macros.svh"
10
11 class test extends uvm_test;
12   `uvm_component_utils(test)
13   ALSU_env AL_env ;
14   shift_reg_env shift_env ;
15   ALSU_config alsu_config_obj_test ;
16   shift_reg_config shift_cfg ;
17   ALSU_reset_sequence reset_sequence ;
18   ALSU_main_sequence main_sequence ;
19
20   function new(string name = "test" , 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     AL_env = ALSU_env::type_id::create("AL_env",this );
27     shift_env = shift_reg_env::type_id::create("shift_env",this );
28     alsu_config_obj_test = ALSU_config::type_id::create("alsu_config_obj_test");
29     shift_cfg = shift_reg_config::type_id::create("shift_cfg");
30     main_sequence = ALSU_main_sequence::type_id::create("main_sequence" );
31     reset_sequence = ALSU_reset_sequence::type_id::create("reset_sequence");
32     if(!uvm_config_db#(virtual ALSU_if)::get (this , "" , "ALSUif" , alsu_config_obj_test.alsu_config_vif ))
33       `uvm_fatal("run_phase" , "test - unable to get the virtual interface ") ;
34     uvm_config_db#(ALSU_config)::set (this , "*" , "CFG" , alsu_config_obj_test ) ;
35
36     if(!uvm_config_db#(virtual shift_reg_if)::get (this , "" , "SHIFT_IF" , shift_cfg.shift_vif ))
37       `uvm_fatal("run_phase" , "test - unable to get the virtual interface ") ;
38     uvm_config_db#(shift_reg_config)::set (this , "*" , "CFG" , shift_cfg ) ;
39     alsu_config_obj_test.is_active = UVM_ACTIVE ;
40     shift_cfg.is_active = UVM_PASSIVE ;
41   endfunction
42
43   task run_phase(uvm_phase phase) ;
44     super.run_phase(phase);
45     phase.raise_objection(this);
46     `uvm_info("run_phase","reset assert" , UVM_LOW)
47     reset_sequence.start(AL_env.agt.sqr);
48     `uvm_info("run_phase","reset deassert" , UVM_LOW)
49
50     `uvm_info("run_phase","stimulus generation started" , UVM_LOW)
51     main_sequence.start(AL_env.agt.sqr);
52     `uvm_info("run_phase","stimulus generation ended" , UVM_LOW)
53     phase.drop_objection(this);
54   endtask:run_phase
55
56 endclass
57 endpackage
```

7- alsu_env

```
1 package ALSU_env_pkg ;
2
3 import ALSU_agent_pkg::*;
4 import ALSU_scoreboard_pkg::*;
5 import ALSU_coverage_pkg::*;
6 import uvm_pkg::* ;
7 `include "uvm_macros.svh"
8
9
10 class ALSU_env extends uvm_env;
11     `uvm_component_utils(ALSU_env)
12
13     ALSU_agent agt ;
14     ALSU_scoreboard sb ;
15     ALSU_coverage cov ;
16
17     function new(string name = "ALSU_env" , uvm_component parent = null );
18         super.new(name ,parent ) ;
19     endfunction
20
21     function void build_phase(uvm_phase phase) ;
22         super.build_phase(phase);
23         agt = ALSU_agent::type_id::create("agt",this );
24         sb = ALSU_scoreboard::type_id::create("sb",this );
25         cov = ALSU_coverage::type_id::create("cov",this );
26     endfunction
27
28
29     function void connect_phase(uvm_phase phase) ;
30         agt.agt_ap.connect(sb.sb_export);
31         agt.agt_ap.connect(cov.cov_export);
32     endfunction
33
34 endclass
35 endpackage
```

8- Shift env

```
C:\Users\> CS > Downloads > Karen Waseem Diploma > session0_Assignmen_extra > ALSO_shift > = shift_reg_env.sv
1 package shift_reg_env_pkg ;
2
3 import shift_reg_agent_pkg::*;
4 import shift_reg_scoreboard_pkg::*;
5 import shift_reg_coverage_pkg::*;
6 import uvm_pkg::* ;
7 `include "uvm_macros.svh"
8
9
10 class shift_reg_env extends uvm_env;
11     `uvm_component_utils(shift_reg_env)
12
13     shift_reg_agent agt ;
14     shift_reg_scoreboard sb ;
15     shift_reg_coverage cov ;
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         | agt = shift_reg_agent::type_id::create("agt",this );
23         | sb = shift_reg_scoreboard::type_id::create("sb",this );
24         | cov = shift_reg_coverage::type_id::create("cov",this );
25     endfunction
26
27     function void connect_phase(uvm_phase phase) ;
28         | agt.agt_ap.connect(sb.sb_export);
29         | agt.agt_ap.connect(cov.cov_export);
30     endfunction
31 endclass
32 endpackage
```


9- reset sequence

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

10- Main sequence

```
1 package ALSU_seq_main_pkg;
2
3 import ALSU_seq_item_pkg::*;
4 import uvm_pkg::* ;
5 `include "uvm_macros.svh"
6
7
8 class ALSU_main_sequence extends uvm_sequence #(ALSU_seq_item);
9     `uvm_object_utils(ALSU_main_sequence)
10
11     ALSU_seq_item seq_item ;
12     function new(string name = "ALSU_main_sequence" );
13         super.new(name) ;
14     endfunction
15     task body;
16         repeat(100000)
17             begin
18                 seq_item = ALSU_seq_item::type_id::create("seq_item");
19                 start_item(seq_item);
20                 assert(seq_item.randomize()) ;
21                 finish_item(seq_item);
22             end
23         endtask
24     endclass
25 endpackage
```

11- ALSU agent

```
1  package ALSU_agent_pkg ;
2
3      import ALSU_driver_pkg::*;
4      import ALSU_sequencer_pkg::*;
5      import ALSU_monitor_pkg::*;
6      import ALSU_config_pkg::*;
7      import ALSU_seq_item_pkg::* ;
8      import uvm_pkg::* ;
9      `include "uvm_macros.svh"
10
11
12      class ALSU_agent extends uvm_agent;
13          `uvm_component_utils(ALSU_agent)
14
15          ALSU_driver  driver ;
16          ALSU_sequencer sqr ;
17          ALSU_monitor mon ;
18          ALSU_config  ALSU_cfg ;
19          uvm_analysis_port #(ALSU_seq_item) agt_ap ;
20
21          function new(string name = "ALSU_agent" , uvm_component parent = null );
22              super.new(name ,parent ) ;
23          endfunction
24
```

```
24
25      function void build_phase(uvm_phase phase) ;
26          super.build_phase(phase);
27          if(!uvm_config_db#(ALSU_config)::get (this , "" , "CFG" , ALSU_cfg ) )
28              `uvm_fatal("build_phase" , "driver - unable to get the virtual interface ") ;
29
30          driver = ALSU_driver::type_id::create("driver",this );
31          sqr = ALSU_sequencer::type_id::create("sqr",this );
32          mon = ALSU_monitor::type_id::create("mon",this );
33          agt_ap = new("agt_ap" , this) ;
34      endfunction
35
36      function void connect_phase(uvm_phase phase) ;
37          driver.alsu_driver_vif = ALSU_cfg.alsu_config_vif ;
38          mon.ALSU_vif = ALSU_cfg.alsu_config_vif ;
39          driver.seq_item_port.connect(sqr.seq_item_export) ;
40          mon.mon_ap.connect(agt_ap);
41      endfunction
42  endclass
43  endpackage
```

12- Shift agent

```
1  package shift_reg_agent_pkg ;
2
3  import shift_reg_driver_pkg::*;
4  import shift_reg_sequencer_pkg::*;
5  import shift_reg_monitor_pkg::*;
6  import shift_reg_config_pkg::*;
7  import shift_reg_seq_item_pkg::* ;
8  import uvm_pkg::* ;
9  `include "uvm_macros.svh"
10
11
12  class shift_reg_agent extends uvm_agent;
13      `uvm_component_utils(shift_reg_agent)
14
15      shift_reg_driver  shift_driver ;
16      shift_reg_sequencer sqr ;
17      shift_reg_monitor mon ;
18      shift_reg_config  shift_cfg ;
19      uvm_analysis_port #(shift_reg_seq_item) agt_ap ;
20
21      function new(string name = "shift_reg_agent" , uvm_component parent = null );
22          | super.new(name ,parent ) ;
23      endfunction
24
25      function void build_phase(uvm_phase phase) ;
26          | super.build_phase(phase);
27          | if(!uvm_config_db#(shift_reg_config)::get (this , "" , "CFG" , shift_cfg ) )
28              | `uvm_fatal("build_phase" , "driver - unable to get the virtual interface ") ;
29          | shift_driver = shift_reg_driver::type_id::create("shift_driver",this );
30          | sqr = shift_reg_sequencer::type_id::create("sqr",this );
31          | mon = shift_reg_monitor::type_id::create("mon",this );
32          | agt_ap = new("agt_ap" , this) ;
33      endfunction
34
35      function void connect_phase(uvm_phase phase) ;
36          | shift_driver.shift_vif = shift_cfg.shift_vif ;
37          | mon.shift_vif = shift_cfg.shift_vif ;
38          | shift_driver.seq_item_port.connect(sqr.seq_item_export) ;
39          | mon.mon_ap.connect(agt_ap);
40      endfunction
41  endclass
42  endpackage
```

13- Shift scoreboard

```
1  package shift_reg_scoreboard_pkg ;
2      import shift_reg_seq_item_pkg::* ;
3      import uvm_pkg::* ;
4      `include "uvm_macros.svh"
5      class shift_reg_scoreboard extends uvm_scoreboard;
6          `uvm_component_utils(shift_reg_scoreboard)
7          uvm_analysis_export #(shift_reg_seq_item) sb_export ;
8          uvm_tlm_analysis_fifo #(shift_reg_seq_item) sb_fifo ;
9          shift_reg_seq_item seq_item_cov ;
10         logic [5:0] dataout_ref ;
11         int error_count = 0 ;
12         int correct_count = 0 ;
13
14         function new(string name = "shift_reg_scoreboard" , uvm_component parent = null ) ;
15             super.new(name ,parent ) ;
16         endfunction
17
18         function void build_phase(uvm_phase phase) ;
19             super.build_phase(phase);
20             sb_export = new("sb_export" , this) ;
21             sb_fifo = new("sb_fifo" , this) ;
22         endfunction
23
24         function void connect_phase(uvm_phase phase) ;
25             sb_export.connect(sb_fifo.analysis_export);
26         endfunction
27
28         task run_phase(uvm_phase phase) ;
29             super.run_phase(phase);
30             forever
31                 begin
32                     sb_fifo.get(seq_item_cov) ;
33                     ref_model(seq_item_cov) ;
34                     if(seq_item_cov.dataout != dataout_ref )
35                         begin
36                             `uvm_error("run_phase" , $sformatf("comparsion failed trasnsaction received by the dut %s shile the reference out %ob"\
37                                 ,seq_item_cov.convert2string , dataout_ref )) ;
38                             error_count++ ;
39                         end
40                     else
41                         begin
42                             `uvm_info("run_phase" ,seq_item_cov.convert2string_stimulus() , UVM_HIGH ) ;
43                             correct_count++ ;
44                         end
45                 end
46         endtask
```

```

47
48     task ref_model(shift_reg_seq_item seq_item_cov);
49         if (seq_item_cov.mode) // rotate
50             if (seq_item_cov.direction) // left
51                 dataout_ref <= {seq_item_cov.datain[4:0], seq_item_cov.datain[5]};
52             else
53                 dataout_ref <= {seq_item_cov.datain[0], seq_item_cov.datain[5:1]};
54         else // shift
55             if (seq_item_cov.direction) // left
56                 dataout_ref <= {seq_item_cov.datain[4:0], seq_item_cov.serial_in};
57             else
58                 dataout_ref <= {seq_item_cov.serial_in, seq_item_cov.datain[5:1]};
59         endtask
60
61     function void report_phase(uvm_phase phase) ;
62         super.report_phase(phase);
63         `uvm_info("report_phase" , $sformatf("total successful %0d " , correct_count ) , UVM_MEDIUM ) ;
64         `uvm_info("report_phase" , $sformatf("total FAILED %0d " , error_count ) , UVM_MEDIUM ) ;
65     endfunction
66 endclass
67 endpackage

```

14- ALSU scoreboard

```

1 package ALSU_scoreboard_pkg ;
2 import ALSU_seq_item_pkg::* ;
3 import uvm_pkg::* ;
4 `include "uvm_macros.svh"
5 class ALSU_scoreboard extends uvm_scoreboard;
6     `uvm_component_utils(ALSU_scoreboard)
7     uvm_analysis_export #(ALSU_seq_item) sb_export ;
8     uvm_tlm_analysis_fifo #(ALSU_seq_item) sb_fifo ;
9     ALSU_seq_item seq_item_cov ;
10    logic [5:0] dataout_ref ;
11    logic [15:0] leds_ref ;
12
13    logic cin_reg, red_op_A_reg, red_op_B_reg, bypass_A_reg, bypass_B_reg, direction_reg, serial_in_reg;
14    logic [2:0] opcode_reg;
15    logic signed [2:0] A_reg, B_reg;
16    logic invalid_red_op, invalid_opcode, invalid;
17    int error_count = 0 ;
18    int correct_count = 0 ;
19    function new(string name = "ALSU_scoreboard" , uvm_component parent = null ) ;
20        super.new(name ,parent ) ;
21    endfunction
22    function void build_phase(uvm_phase phase) ;
23        super.build_phase(phase);
24        sb_export = new("sb_export" , this) ;
25        sb_fifo = new("sb_fifo" , this) ;
26    endfunction
27
28    function void connect_phase(uvm_phase phase) ;
29        sb_export.connect(sb_fifo.analysis_export);
30    endfunction
31

```

```

32 task run_phase(uvm_phase phase) ;
33     super.run_phase(phase);
34     forever
35     begin
36         sb_fifo.get(seq_item_cov) ;
37         ref_model(seq_item_cov) ;
38         if(seq_item_cov.out != dataout_ref && seq_item_cov.leds != leds_ref )
39         begin
40             `uvm_error("run_phase" , $sformatf("comparsion failed trasnsaction received by the dut %s shile \
41             | the reference out %ob" ,seq_item_cov.convert2string , dataout_ref )) ;
42             error_count++ ;
43         end
44         else
45         begin
46             `uvm_info("run_phase" ,seq_item_cov.convert2string_stimulus() , UVM_HIGH ) ;
47             correct_count++ ;
48         end
49     end
50 endtask

```

```

52 task ref_model(ASLU_seq_item seq_item_cov);
53     invalid_red_op = (red_op_A_reg | red_op_B_reg) & (opcode_reg[1] | opcode_reg[2]);
54     invalid_opcode = opcode_reg[1] & opcode_reg[2];
55     invalid = invalid_red_op | invalid_opcode;
56     fork
57     begin
58         if(seq_item_cov.rst) begin
59             cin_reg <= 0;
60             red_op_B_reg <= 0;
61             red_op_A_reg <= 0;
62             bypass_B_reg <= 0;
63             bypass_A_reg <= 0;
64             direction_reg <= 0;
65             serial_in_reg <= 0;
66             opcode_reg <= 0;
67             A_reg <= 0;
68             B_reg <= 0;
69         end else begin
70             cin_reg <= seq_item_cov.cin;
71             red_op_B_reg <= seq_item_cov.red_op_B;
72             red_op_A_reg <= seq_item_cov.red_op_A;
73             bypass_B_reg <= seq_item_cov.bypass_B;
74             bypass_A_reg <= seq_item_cov.bypass_A;
75             direction_reg <= seq_item_cov.direction;
76             serial_in_reg <= seq_item_cov.serial_in;
77             opcode_reg <= seq_item_cov.opcode;
78             A_reg <= seq_item_cov.A;
79             B_reg <= seq_item_cov.B;
80         end
81     end
82

```

```

82
83     begin
84         if(seq_item_cov.rst) begin
85             dataout_ref <= 0;
86         end
87         else begin
88             if (invalid)
89                 dataout_ref <= 0;
90             else if (bypass_A_reg && bypass_B_reg)
91                 dataout_ref <= A_reg;
92             else if (bypass_A_reg)
93                 dataout_ref <= A_reg;
94             else if (bypass_B_reg)
95                 dataout_ref <= B_reg;
96             else begin
97                 case (opcode_reg)
98                     3'h0: begin
99                         if (red_op_A_reg && red_op_B_reg)
100                             dataout_ref <= |A_reg;
101                         else if (red_op_A_reg)
102                             dataout_ref <= |A_reg;
103                         else if (red_op_B_reg)
104                             dataout_ref <= |B_reg;
105                         else
106                             dataout_ref <= A_reg | B_reg;
107                     end
108                     3'h1: begin
109                         if (red_op_A_reg && red_op_B_reg)
110                             dataout_ref <= ^A_reg;
111                             dataout_ref <= ^A_reg;
112                             dataout_ref <= ^A_reg;
113                             else if (red_op_B_reg)
114                                 dataout_ref <= ^B_reg;
115                             else
116                                 dataout_ref <= A_reg ^ B_reg;
117                             end
118                             3'h2:begin
119                                 dataout_ref <= A_reg + B_reg + cin_reg ;
120                                 end
121                             3'h3: dataout_ref <= A_reg * B_reg;
122                             3'h4: begin
123                                 if (direction_reg)
124                                     dataout_ref <= {dataout_ref[4:0], serial_in_reg};
125                                 else
126                                     dataout_ref <= {serial_in_reg, dataout_ref[5:1]};
127                                 end
128                             3'h5: begin
129                                 if (direction_reg)
130                                     dataout_ref <= {dataout_ref[4:0], dataout_ref[5]};
131                                 else
132                                     dataout_ref <= {dataout_ref[0], dataout_ref[5:1]};
133                                 end
134                             default : dataout_ref <= 0 ;
135                             endcase
136                         end
137                     end
138                 end
139             end

```



```

140
141         join
142         if(seq_item_cov.rst) begin
143             leds_ref <= 0;
144         end else begin
145             if (invalid)
146                 leds_ref <= ~leds_ref;
147             else
148                 leds_ref <= 0;
149         end
150     endtask
151
152     function void report_phase(uvm_phase phase) ;
153     super.report_phase(phase);
154     `uvm_info("report_phase", $sformatf("total successful %0d ", correct_count ), UVM_MEDIUM ) ;
155     `uvm_info("report_phase", $sformatf("total FAILED %0d ", error_count ), UVM_MEDIUM ) ;
156     endfunction
157 endclass
158 endpackage

```

15- ALSU coverage

```

1 package ALSU_coverage_pkg ;
2
3 import ALSU_seq_item_pkg::* ;
4 import uvm_pkg::* ;
5 `include "uvm_macros.svh"
6
7
8 class ALSU_coverage extends uvm_component;
9     `uvm_component_utils(ALSU_coverage)
10
11     uvm_analysis_export #(ALSU_seq_item) cov_export ;
12     uvm_tlm_analysis_fifo #(ALSU_seq_item) cov_fifo ;
13     ALSU_seq_item seq_item_cov ;
14     typedef enum logic [2:0] {OR , XOR , ADD , MULT , SHIFT , ROTATE , INVALID_6 , INVALID_7 } reg_e ;
15     typedef enum {Or , Xor , Add , Mult , Shift , Rotate} opcode_valid_e ;
16     localparam MAXPOS = 3 ;
17     localparam MAXNEG = -4 ;
18     localparam zero = 0 ;
19     covergroup cvr_gp ;
20         A_cp : coverpoint seq_item_cov.A {
21             bins A_data_0 = {0} ;
22             bins A_data_max = {MAXPOS} ;
23             bins A_data_min = {MAXNEG} ;
24             bins A_data_default = default ;
25             bins A_data_walkingones[] = {1, 2, -4} iff (seq_item_cov.red_op_A && !seq_item_cov.red_op_B);
26         }
27
28         B_cp : coverpoint seq_item_cov.B {
29             bins B_data_0 = {0} ;
30             bins B_data_max = {MAXPOS} ;
31             bins B_data_min = {MAXNEG} ;
32             bins B_data_default = default ;
33             bins B_data_walkingones[] = {1, 2, -4} iff (!seq_item_cov.red_op_A && seq_item_cov.red_op_B);
34         }

```

```

35
36     ALU_cp : coverpoint seq_item_cov.opcode {
37         bins Bins_shift[] = {SHIFT , ROTATE} ;
38         bins Bins_arith[] = {ADD , MULT} ;
39         bins Bins_bitwise[] = {OR , XOR} ;
40         illegal_bins Bins_invalid = {INVALID_6 , INVALID_7} ;
41     }
42
43     cin_cp : coverpoint seq_item_cov.cin {
44         bins cin_data = {0 , 1} ;
45     }
46
47     direction_cp : coverpoint seq_item_cov.direction {
48         bins direction_data = {0 , 1} ;
49     }
50
51     serial_in_cp : coverpoint seq_item_cov.serial_in {
52         bins serial_in_data = {0 , 1} ;
53     }
54
55     red_op_A_cp : coverpoint seq_item_cov.red_op_A {
56         bins red_op_A_LOW_data = {0} ;
57         bins red_op_A_HIGH_data = {1} ;
58     }
59     red_op_B_cp : coverpoint seq_item_cov.red_op_B {
60         bins red_op_B_LOW_data = {0} ;
61         bins red_op_B_HIGH_data = {1} ;
62     }
63
64
65
66     add_mult_cp1 : cross A_cp , B_cp , ALU_cp
67     {
68
69         bins zero_A_add = binsof(ALU_cp.Bins_arith) && binsof(A_cp.A_data_0) && binsof(B_cp.B_data_0) ;
70         bins max_pos_add = binsof(ALU_cp.Bins_arith) && binsof(A_cp.A_data_max) && binsof(B_cp.B_data_max) ;
71         option.cross_auto_bin_max = 0 ;
72     }
73
74
75
76     opcode_cp2 : cross cin_cp , direction_cp , serial_in_cp , ALU_cp , red_op_A_cp , red_op_B_cp
77     {
78
79         bins cin_add = binsof(cin_cp.cin_data) && binsof(ALU_cp.Bins_arith) intersect {ADD} ;
80         bins serialin_shift = binsof(serial_in_cp.serial_in_data) && binsof(ALU_cp) intersect {SHIFT} ;
81         bins direction_shift_rota = binsof(direction_cp) && binsof(ALU_cp.Bins_shift) ;
82         option.cross_auto_bin_max = 0 ;
83     }
84
85
86
87     or_xor_cp3 : cross A_cp , B_cp , ALU_cp , red_op_A_cp , red_op_B_cp
88     {
89
90         bins or_xor_data_A = binsof(ALU_cp.Bins_bitwise) && binsof(A_cp.A_data_walkingones) && binsof(B_cp.B_data_0) \
91         && binsof(red_op_B_cp.red_op_B_LOW_data) && binsof(red_op_A_cp.red_op_A_HIGH_data) ;
92         bins or_xor_data_B = binsof(ALU_cp.Bins_bitwise) && binsof(B_cp.B_data_walkingones) && binsof(A_cp.A_data_0) \
93         && binsof(red_op_A_cp.red_op_A_LOW_data) && binsof(red_op_B_cp.red_op_B_HIGH_data) ;
94         option.cross_auto_bin_max = 0 ;
95     }

```

```

96     INVALID_cp4 : cross  ALU_cp , red_op_A_cp , red_op_B_cp
97     {
98         bins Bins_shift_data = binsof(ALU_cp.Bins_shift) && ( binsof(red_op_B_cp.red_op_B_HIGH_data) || binsof(red_op_A_cp.red_op_A_HIGH_data) ) ;
99         bins Bins_arith_data = binsof(ALU_cp.Bins_arith) && ( binsof(red_op_B_cp.red_op_B_HIGH_data) || binsof(red_op_A_cp.red_op_A_HIGH_data) ) ;
100         option.cross_auto_bin_max = 0 ;
101     }
102
103
104 endgroup
105 function new(string name = "ALSU_sequencer" , uvm_component parent = null );
106     super.new(name ,parent ) ;
107     cvr_gp=new();
108 endfunction
109
110 function void build_phase(uvm_phase phase) ;
111     super.build_phase(phase);
112     cov_export = new("cov_export" , this) ;
113     cov_fifo = new("cov_fifo" , this) ;
114
115 endfunction
116
117 function void connect_phase(uvm_phase phase) ;
118     super.connect_phase(phase);
119     cov_export.connect(cov_fifo.analysis_export);
120 endfunction
121
122 task run_phase(uvm_phase phase) ;
123     super.run_phase(phase);
124     forever
125     begin
126         cov_fifo.get(seq_item_cov) ;
127         cvr_gp.sample();
128     end
129 endtask
130 endclass
131 endpackage

```

16- ALSU sequencer

```

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

```

17- Shift sequencer

```

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

```

18- Shift monitor

```

1 package shift_reg_monitor_pkg ;
2 import uvm_pkg::* ;
3 import shift_reg_seq_item_pkg::* ;
4 import shared_pkg::* ;
5 `include "uvm_macros.svh"
6 class shift_reg_monitor extends uvm_monitor ;
7     `uvm_component_utils(shift_reg_monitor)
8     virtual shift_reg_if shift_vif ;
9     shift_reg_seq_item seq_item ;
10    uvm_analysis_port #(shift_reg_seq_item) mon_ap ;
11    function new(string name = "shift_reg_monitor" , 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_ap = new("mon_ap" , this) ;
17    endfunction
18    task run_phase(uvm_phase phase) ;
19        super.run_phase(phase);
20        forever
21        begin
22            seq_item = shift_reg_seq_item::type_id::create("seq_item");
23            #2 ;
24            seq_item.serial_in = shift_vif.serial_in ;
25            seq_item.direction = direction_e'(shift_vif.direction) ;
26            seq_item.mode = mode_e'(shift_vif.mode) ;
27            seq_item.datain = shift_vif.datain ;
28            //seq_item.dataout = shift_vif.dataout ;
29            mon_ap.write(seq_item);
30            `uvm_info("run_phase" ,seq_item.convert2string_stimulus() , UVM_HIGH )
31        end
32    endtask
33 endclass
34 endpackage

```

19- ALSU monitor

```

1  package ALSU_monitor_pkg ;
2
3  import uvm_pkg::* ;
4  import ALSU_seq_item_pkg::* ;
5  `include "uvm_macros.svh"
6
7  class ALSU_monitor extends uvm_monitor ;
8      `uvm_component_utils(ALSU_monitor)
9
10     virtual ALSU_if ALSU_vif ;
11     ALSU_seq_item seq_item ;
12
13     uvm_analysis_port #(ALSU_seq_item) mon_ap ;
14
15     function new(string name = "ALSU_monitor" , 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         mon_ap = new("mon_ap" , this) ;
22     endfunction
23
24     task run_phase(uvm_phase phase) ;
25         super.run_phase(phase);
26         forever
27             begin
28                 seq_item = ALSU_seq_item::type_id::create("seq_item");
29
30                 @(negedge ALSU_vif.clk ) ;
31                 seq_item.serial_in = ALSU_vif.serial_in ;
32                 seq_item.red_op_A = ALSU_vif.red_op_A ;
33                 seq_item.red_op_B = ALSU_vif.red_op_B ;
34                 seq_item.bypass_A = ALSU_vif.bypass_A ;
35                 seq_item.bypass_B = ALSU_vif.bypass_B ;
36                 seq_item.direction = ALSU_vif.direction ;
37                 seq_item.serial_in = ALSU_vif.serial_in ;
38                 seq_item.opcode = ALSU_vif.opcode ;
39                 seq_item.A = ALSU_vif.A ;
40                 seq_item.B = ALSU_vif.B ;
41                 seq_item.cin = ALSU_vif.cin ;
42                 mon_ap.write(seq_item);
43                 `uvm_info("run_phase" ,seq_item.convert2string_stimulus() , UVM_HIGH )
44
45             end
46         endtask
47     endclass
48 endpackage
49

```

20- alsu_driver

```

1  package ALSU_driver_pkg ;
2
3  import uvm_pkg::* ;
4  import ALSU_seq_item_pkg::* ;
5  `include "uvm_macros.svh"
6
7
8  class ALSU_driver extends uvm_driver #(ALSU_seq_item);
9      `uvm_component_utils(ALSU_driver)
10     virtual ALSU_if alsu_driver_vif ;
11     ALSU_seq_item seq_item ;
12
13     function new(string name = "ALSU_driver" , uvm_component parent = null ) ;
14         super.new(name ,parent ) ;
15     endfunction
16
17     task run_phase(uvm_phase phase) ;
18         super.run_phase(phase);
19         alsu_driver_vif.rst = 1 ;
20         alsu_driver_vif.red_op_A = 0 ;
21         alsu_driver_vif.red_op_B = 0 ;
22         alsu_driver_vif.bypass_A = 0 ;
23         alsu_driver_vif.bypass_B = 0 ;
24         alsu_driver_vif.direction = 0 ;
25         alsu_driver_vif.serial_in = 0 ;
26         alsu_driver_vif.opcode = 0 ;
27         alsu_driver_vif.A = 0 ;
28         alsu_driver_vif.B = 0 ;
29         alsu_driver_vif.cin = 0 ;
30
31         @(negedge alsu_driver_vif.clk ) ;
32         alsu_driver_vif.rst = 0 ;
33
34         forever
35         begin
36             seq_item = ALSU_seq_item::type_id::create("seq_item");
37             seq_item_port.get_next_item(seq_item);
38             @(negedge alsu_driver_vif.clk ) ;
39             alsu_driver_vif.serial_in = seq_item.serial_in ;
40             alsu_driver_vif.red_op_A = seq_item.red_op_A ;
41             alsu_driver_vif.red_op_B = seq_item.red_op_B ;
42             alsu_driver_vif.bypass_A = seq_item.bypass_A ;
43             alsu_driver_vif.bypass_B = seq_item.bypass_B ;
44             alsu_driver_vif.direction = seq_item.direction ;
45             alsu_driver_vif.serial_in = seq_item.serial_in ;
46             alsu_driver_vif.opcode = seq_item.opcode ;
47             alsu_driver_vif.A = seq_item.A ;
48             alsu_driver_vif.B = seq_item.B ;
49             alsu_driver_vif.cin = seq_item.cin ;
50             seq_item_port.item_done();
51             `uvm_info("run_phase" ,seq_item.convert2string_stimulus() , UVM_HIGH )
52         end
53     endtask
54 endclass
endpackage

```

21- Shift driver

```
1 package shift_reg_driver_pkg ;
2 import uvm_pkg::* ;
3 import shift_reg_seq_item_pkg::* ;
4 `include "uvm_macros.svh"
5 class shift_reg_driver extends uvm_driver #(shift_reg_seq_item);
6     `uvm_component_utils(shift_reg_driver)
7     virtual shift_reg_if shift_vif ;
8     shift_reg_seq_item seq_item ;
9     function new(string name = "shift_reg_driver" , uvm_component parent = null );
10         super.new(name ,parent ) ;
11     endfunction
12     task run_phase(uvm_phase phase) ;
13         super.run_phase(phase);
14         shift_vif.serial_in = 0 ;
15         shift_vif.direction = 0 ;
16         shift_vif.mode = 0 ;
17         shift_vif.datain = 0 ;
18         #2 ;
19         forever
20             begin
21                 seq_item = shift_reg_seq_item::type_id::create("seq_item");
22                 seq_item_port.get_next_item(seq_item);
23                 #2 ;
24                 shift_vif.serial_in = seq_item.serial_in ;
25                 shift_vif.direction = seq_item.direction ;
26                 shift_vif.mode = seq_item.mode ;
27                 shift_vif.datain = seq_item.datain ;
28                 //@(negedge shift_vif.clk ) ;
29                 seq_item_port.item_done();
30                 `uvm_info("run_phase" ,seq_item.convert2string_stimulus() , UVM_HIGH )
31             end
32         endtask
33     endclass
34 endpackage
35
```

22- ALSU seq_item

```

1 package ALSU_seq_item_pkg ;
2 import uvm_pkg::* ;
3 `include "uvm_macros.svh"
4
5 class ALSU_seq_item extends uvm_sequence_item;
6   `uvm_object_utils(ALSU_seq_item)
7   typedef enum logic [2:0] {OR , XOR , ADD , MULT , SHIFT , ROTATE , INVALID_6 , INVALID_7 } reg_e ;
8   typedef enum {Or , Xor , Add , Mult , Shift , Rotate} opcode_valid_e ;
9   localparam MAXPOS = 3 ;
10  localparam MAXNEG = -4 ;
11  localparam zero = 0 ;
12  rand bit rst, cin, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
13  rand bit [2:0] opcode;
14  rand bit signed [2:0] A, B;
15  logic [15:0] leds;
16  logic signed [5:0] out;
17
18  constraint rst_n {rst dist {0:/99 , 1:/1 } ; }
19
20  constraint input_A {
21    if ( (opcode == ADD ) || (opcode == MULT ) )
22    {
23      A dist { MAXPOS:/25 , MAXNEG:/25 , zero:/25 , {3'b010 , 3'b001 , 3'b111 , 3'b110 , 3'b101}:/25 } ;
24    }
25    else if (((opcode == XOR ) || (opcode == OR )) && (red_op_A == 1) && (red_op_B == 0) )
26    {
27      B == 0 ;
28      A dist {3'b001:/30 , 3'b010:/30 , 3'b100:/30 , 3'b000:/5 , 3'b011:/5 , 3'b101:/5 , 3'b110:/5 , 3'b111:/5 } ;
29    }
30    else
31    {
32      A inside { [MAXNEG : MAXPOS] } ;
33    }
34  }
35
36  constraint input_B {
37    if ( (opcode == ADD ) || (opcode == MULT ) )
38    {
39      B dist { MAXPOS:/25 , MAXNEG:/25 , zero:/25 , {3'b010 , 3'b001 , 3'b111 , 3'b110 , 3'b101}:/25 } ;
40    }
41    else if (((opcode == XOR ) || (opcode == OR )) && (red_op_B == 1) && (red_op_A == 0) )
42    {
43      A == 0 ;
44      B dist {3'b001:/30 , 3'b010:/30 , 3'b100:/30 , 3'b000:/5 , 3'b011:/5 , 3'b101:/5 , 3'b110:/5 , 3'b111:/5 } ;
45    }
46    else
47    {
48      B inside { [MAXNEG : MAXPOS] } ;
49    }
50  }
51
52  constraint input_opcode {opcode dist {[0:3]:/45 , [4:5]:/50 , [6:7]:/1 } ; }
53
54  constraint input_bypass_A {bypass_A dist {1:=90 , 0:=10 } ; }
55  constraint input_bypass_B {bypass_B dist {1:=90 , 0:=10 } ; }
56
57  constraint input_red_op_A {red_op_A dist {1:/10 , 0:/90 } ; }
58  constraint input_red_op_B {red_op_B dist {1:/10 , 0:/90 } ; }
59
60
61  function new(string name = "ALSU_seq_item" );
62    super.new(name) ;
63  endfunction
64
65  function string convert2string();
66  return $sformatf ("%s , rst = %0d , serial_in = %0d , direction = %0d , cin = %0d , red_op_A = %0d , red_op_B = %0d , bypass_A = %0d \
67  , bypass_B = %0d , opcode = %0d , A = %0d , B = %0d , leds = %0d , out = %0d" , super.convert2string , rst , serial_in , \
68  direction , cin , red_op_A , red_op_B , bypass_A , bypass_B , opcode , A , B , leds , out ) ;
69  endfunction
70
71  function string convert2string_stimulus();
72  return $sformatf ("rst = %0d , serial_in = %0d , direction = %0d , cin = %0d , red_op_A = %0d , red_op_B = %0d , bypass_A = %0d \
73  , bypass_B = %0d , opcode = %0d , A = %0d , B = %0d " , rst , serial_in , direction , cin , red_op_A , red_op_B \
74  , bypass_A , bypass_B , opcode , A , B ) ;
75  endfunction
76
77  endclass
78 endpackage

```


23- Shift seq_item

```
1 package shift_reg_seq_item_pkg ;
2 import uvm_pkg::* ;
3 import shared_pkg::* ;
4 `include "uvm_macros.svh"
5
6 class shift_reg_seq_item extends uvm_sequence_item;
7     `uvm_object_utils(shift_reg_seq_item)
8
9
10     logic serial_in;
11     logic direction ;
12     logic mode;
13     logic [5:0] datain ;
14     logic [5:0] dataout;
15
16
17
18     function new(string name = "shift_reg_seq_item" );
19         super.new(name ) ;
20     endfunction
21
22     function string convert2string();
23 return $sformatf ("%s , serial_in = %0d , direction = %0d , mode = %0d , datain = %0d , dataout = %0d\"
24 , super.convert2string , serial_in , direction , mode , datain , dataout ) ;
25     endfunction
26
27     function string convert2string_stimulus();
28         return $sformatf (" serial_in = %0d , direction = %0d , mode = %0d , datain = %0d " , serial_in , direction , mode , datain ) ;
29     endfunction
30
31 endclass
32 endpackage
```

24- alsu_config_obj

```
1 package ALSU_config_pkg ;
2
3     import uvm_pkg::* ;
4     `include "uvm_macros.svh"
5
6     class ALSU_config extends uvm_object;
7         `uvm_object_utils(ALSU_config)
8
9         virtual ALSU_if alsu_config_vif ;
10         function new(string name = "ALSU_config");
11             super.new(name) ;
12         endfunction
13     endclass
14 endpackage
```

25- Shift config

```

1  package shift_reg_config_pkg ;
2
3      import uvm_pkg::* ;
4      `include "uvm_macros.svh"
5
6
7      class shift_reg_config extends uvm_object;
8          `uvm_object_utils(shift_reg_config)
9
10         virtual shift_reg_if shift_vif ;
11         uvm_active_passive_enum is_active ;
12         function new(string name = "shift_reg_config");
13             super.new(name) ;
14         endfunction
15     endclass
16 endpackage

```

26- Do file

```

1  vlib work
2  vlog *v +cover
3  vsim -voptargs=+acc work.ALSU_shift_top -classdebug -uvmcontrol=all -cover
4  add wave /ALSU_shift_top/ALSUif/*
5  coverage save top.ucdb -onexit
6  run -all
7
8  quit -sim
9  vccover report top.ucdb -all -details -output coverage.txt

```

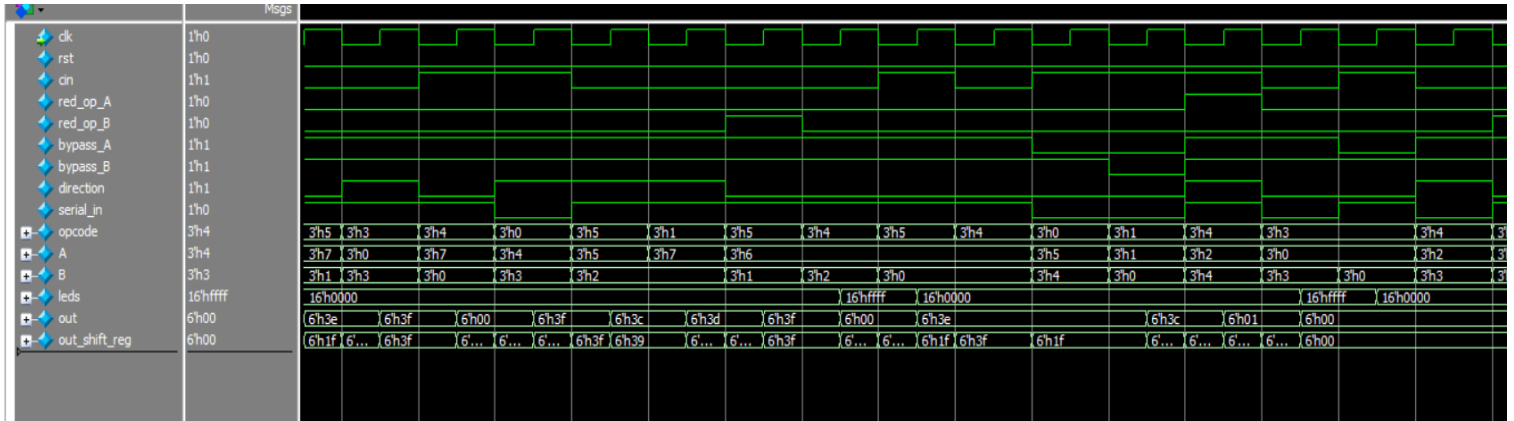
27- Transcript

```

# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_object_utils.svh(1268) @ 200004: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO ALSU_scoreboard.sv(160) @ 200004: uvm_test_top.ALSU_env.sb [report_phase] total successful 100002
# UVM_INFO ALSU_scoreboard.sv(161) @ 200004: uvm_test_top.ALSU_env.sb [report_phase] total FAILED 0
# UVM_INFO shift_reg_scoreboard.sv(67) @ 200004: uvm_test_top.shift_env.sb [report_phase] total successful 100002
# UVM_INFO shift_reg_scoreboard.sv(68) @ 200004: uvm_test_top.shift_env.sb [report_phase] total FAILED 0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO : 12
# UVM_WARNING : 0
# UVM_ERROR : 0
# UVM_FATAL : 0
#
# ** Report counts by id
# [Questa UVM] 2
# [RNTST] 1
# [TEST_DONE] 1
# [report_phase] 4
# [run_phase] 4
#
# ** Note: $finish : C:/questasim64_10.4c/win64/./verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)

```

28- Waveform



29- Assertion

/ALSU_shift_top/d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge ALSUif.clk) disa...	✓
/ALSU_shift_top/d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge ALSUif.clk) disa...	✓
/ALSU_shift_top/d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge ALSUif.clk) disa...	✓
/ALSU_shift_top/d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge ALSUif.clk) disa...	✓
/ALSU_shift_top/d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge ALSUif.clk) disa...	✓
/ALSU_shift_top/d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge ALSUif.clk) disa...	✓
/ALSU_shift_top/d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge ALSUif.clk) disa...	✓
/ALSU_shift_top/d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge ALSUif.clk) disa...	✓
/ALSU_shift_top/d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge ALSUif.clk) disa...	✓
/ALSU_shift_top/d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge ALSUif.clk) disa...	✓
/ALSU_shift_top/d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge ALSUif.clk) disa...	✓
/ALSU_shift_top/d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge ALSUif.clk) disa...	✓
/ALSU_shift_top/d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge ALSUif.clk) disa...	✓
/ALSU_shift_top/d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge ALSUif.clk) disa...	✓
/ALSU_shift_top/d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge ALSUif.clk) disa...	✓
/ALSU_shift_top/d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge ALSUif.clk) disa...	✓
/ALSU_shift_top/d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge ALSUif.clk) disa...	✓
/ALSU_shift_top/d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge ALSUif.clk) disa...	✓
/ALSU_shift_top/d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge ALSUif.clk) disa...	✓
/ALSU_shift_top/d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge ALSUif.clk) disa...	✓
/ALSU_shift_top/d...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge ALSUif.clk) disa...	✓

30- Assertion coverage

/ALSU_shift_top/d...	SVA	✓	Off	10	1	Unli...	1	100%	✓	0	0	0 ns	0
/ALSU_shift_top/d...	SVA	✓	Off	8	1	Unli...	1	100%	✓	0	0	0 ns	0
/ALSU_shift_top/d...	SVA	✓	Off	97	1	Unli...	1	100%	✓	0	0	0 ns	0
/ALSU_shift_top/d...	SVA	✓	Off	9	1	Unli...	1	100%	✓	0	0	0 ns	0
/ALSU_shift_top/d...	SVA	✓	Off	6	1	Unli...	1	100%	✓	0	0	0 ns	0
/ALSU_shift_top/d...	SVA	✓	Off	92	1	Unli...	1	100%	✓	0	0	0 ns	0
/ALSU_shift_top/d...	SVA	✓	Off	97	1	Unli...	1	100%	✓	0	0	0 ns	0
/ALSU_shift_top/d...	SVA	✓	Off	92	1	Unli...	1	100%	✓	0	0	0 ns	0
/ALSU_shift_top/d...	SVA	✓	Off	98	1	Unli...	1	100%	✓	0	0	0 ns	0
/ALSU_shift_top/d...	SVA	✓	Off	105	1	Unli...	1	100%	✓	0	0	0 ns	0
/ALSU_shift_top/d...	SVA	✓	Off	96	1	Unli...	1	100%	✓	0	0	0 ns	0
/ALSU_shift_top/d...	SVA	✓	Off	115	1	Unli...	1	100%	✓	0	0	0 ns	0
/ALSU_shift_top/d...	SVA	✓	Off	525	1	Unli...	1	100%	✓	0	0	0 ns	0
/ALSU_shift_top/d...	SVA	✓	Off	509	1	Unli...	1	100%	✓	0	0	0 ns	0
/ALSU_shift_top/d...	SVA	✓	Off	18	1	Unli...	1	100%	✓	0	0	0 ns	0
/ALSU_shift_top/d...	SVA	✓	Off	24	1	Unli...	1	100%	✓	0	0	0 ns	0
/ALSU_shift_top/d...	SVA	✓	Off	29	1	Unli...	1	100%	✓	0	0	0 ns	0
/ALSU_shift_top/d...	SVA	✓	Off	30	1	Unli...	1	100%	✓	0	0	0 ns	0
/ALSU_shift_top/d...	SVA	✓	Off	17007	1	Unli...	1	100%	✓	0	0	0 ns	0
/ALSU_shift_top/d...	SVA	✓	Off	1730	1	Unli...	1	100%	✓	0	0	0 ns	0

#part2

31- CODE Design

```
3  module ALSU(A, B, cin, serial_in, red_op_A, red_op_B, opcode, bypass_A, bypass_B, clk, rst, direction, leds, out);
4  parameter INPUT_PRIORITY = "A";
5  parameter FULL_ADDER = "ON";
6  input clk, rst, cin, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
7  input [2:0] opcode;
8  input signed [2:0] A, B;    // first bug [ ] -> we must put it signed
9  output reg [15:0] leds;
10 output reg signed [5:0] out; // second bug [ ] -> we must put it signed
11 reg cin_reg, red_op_A_reg, red_op_B_reg, bypass_A_reg, bypass_B_reg, direction_reg, serial_in_reg;
12 reg [2:0] opcode_reg;
13 reg signed [2:0] A_reg, B_reg;
14 wire invalid_red_op, invalid_opcode, invalid;
15 //Invalid handling
16 assign invalid_red_op = (red_op_A_reg | red_op_B_reg) & (opcode_reg[1] | opcode_reg[2]);
17 assign invalid_opcode = opcode_reg[1] & opcode_reg[2];
18 assign invalid = invalid_red_op | invalid_opcode;
19 //Registering input signals
20 always @(posedge clk or posedge rst) begin
21     if(rst) begin
22         cin_reg <= 0;
23         red_op_B_reg <= 0;
24         red_op_A_reg <= 0;
25         bypass_B_reg <= 0;
26         bypass_A_reg <= 0;
27         direction_reg <= 0;
28         serial_in_reg <= 0;
29         opcode_reg <= 0;
30         A_reg <= 0;
31         B_reg <= 0;
32     end else begin
33         cin_reg <= cin;
34         red_op_B_reg <= red_op_B;
35         red_op_A_reg <= red_op_A;
36         bypass_B_reg <= bypass_B;
37         bypass_A_reg <= bypass_A;
38         direction_reg <= direction;
39         serial_in_reg <= serial_in;
40         opcode_reg <= opcode;
41         A_reg <= A;
42         B_reg <= B;
43     end
44 end
45 //leds output blinking
46 always @(posedge clk or posedge rst) begin
47     if(rst) begin
48         leds <= 0;
49     end else begin
50         if (invalid)
51             leds <= ~leds;
52         else
53             leds <= 0;
54     end
55 end
56 //ALSU output processing
```

```

57 always @(posedge clk or posedge rst) begin
58     if(rst) begin
59         out <= 0;
60     end
61     else begin
62         if (invalid)
63             out <= 0;
64         else if (bypass_A_reg && bypass_B_reg)
65             out <= (INPUT_PRIORITY == "A")? A_reg: B_reg;
66         else if (bypass_A_reg)
67             out <= A_reg;
68         else if (bypass_B_reg)
69             out <= B_reg;
70         else begin
71             case (opcode_reg) // third bug is to used the opcode_reg not the opcode
72                 3'h0: begin
73                     if (red_op_A_reg && red_op_B_reg)
74                         out <= (INPUT_PRIORITY == "A")? |A_reg: |B_reg; // third bug is to replace AND with OR
75                     else if (red_op_A_reg)
76                         out <= |A_reg;
77                     else if (red_op_B_reg)
78                         out <= |B_reg;
79                     else
80                         out <= A_reg | B_reg;
81                 end
82                 3'h1: begin
83                     if (red_op_A_reg && red_op_B_reg)
84                         out <= (INPUT_PRIORITY == "A")? ^A_reg: ^B_reg; // fourth bug is to replace OR with XOR
85                     else if (red_op_A_reg)
86                         out <= ^A_reg;
87                     else if (red_op_B_reg)
88                         out <= ^B_reg;
89                     else
90                         out <= A_reg ^ B_reg;
91                 end
92                 3'h2: begin
93                     if (FULL_ADDER == "ON") // fifth bug to add Cin operation in case of full adde
94                         out <= A_reg + B_reg + cin_reg ;
95                     else
96                         out <= A_reg + B_reg ;
97                     end
98                 3'h3: out <= A_reg * B_reg;
99                 3'h4: begin
100                     if (direction_reg)
101                         out <= {out[4:0], serial_in_reg};
102                     else
103                         out <= {serial_in_reg, out[5:1]};
104                     end
105                 3'h5: begin
106                     if (direction_reg)
107                         out <= {out[4:0], out[5]};
108                     else
109                         out <= {out[0], out[5:1]};
110                     end
111                 default : out <= 0 ;
112             endcase
113         end
114     end
115 end
116
117 endmodule

```

32- Interface

```
1 interface ALSU_if (clk);
2   input clk ;
3   logic rst, cin, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
4   logic [2:0] opcode;
5   logic signed [2:0] A, B;
6   logic [15:0] leds;
7   logic signed [5:0] out;
8
9   modport DUT (input clk , rst, cin, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in , opcode , A , B , output leds , out ) ;
10
11 endinterface
```

33- top module

```
C:\> Users > CS > Downloads > Kareem Wasem Diploma > session6_Assignmen > ALSU > ALSU_top.sv
1  import ALSU_test_pkg::*;
2  import uvm_pkg::* ;
3  `include "uvm_macros.svh"
4
5
6
7  module ALSU_top();
8  bit clk ;
9
10  initial
11  begin
12      clk = 0 ;
13      forever
14      #1 clk = ~clk ;
15  end
16  ALSU_if ALSUif (clk) ;
17  ALSU dut ( ALSUif.A, ALSUif.B, ALSUif.cin, ALSUif.serial_in, ALSUif.red_op_A, ALSUif.red_op_B, \
18  ALSUif.opcode, ALSUif.bypass_A, ALSUif.bypass_B, ALSUif.clk, ALSUif.rst, ALSUif.direction, ALSUif.leds, ALSUif.out) ;
19  bind ALSU SVA sva(ALSUif.DUT) ;
20  initial
21  begin
22      uvm_config_db#(virtual ALSU_if)::set(null , "uvm_test_top" , "ALSUif" , ALSUif );
23      run_test("ALSU_test");
24  end
25
26
27  endmodule
```

34- alsu_test

```

1 package ALSU_test_pkg ;
2
3 import ALSU_env_pkg::*;
4 import ALSU_config_pkg::*;
5 import ALSU_seq_reset_pkg::*;
6 import ALSU_seq_main_pkg::*;
7 import ALSU_seq_item_pkg::* ;
8 import alsu_seq_item_valid_invalid_pkg::* ;
9 import uvm_pkg::* ;
10 `include "uvm_macros.svh"
11
12
13 class ALSU_test extends uvm_test;
14     `uvm_component_utils(ALSU_test)
15     ALSU_env env ;
16     ALSU_config alsu_config_obj_test ;
17     ALSU_reset_sequence reset_sequence ;
18     ALSU_main_sequence main_sequence ;
19
20     function new(string name = "ALSU_test" , 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         env = ALSU_env::type_id::create("env",this );
27         alsu_config_obj_test = ALSU_config::type_id::create("alsu_config_obj_test");
28         main_sequence = ALSU_main_sequence::type_id::create("main_sequence" );
29         reset_sequence = ALSU_reset_sequence::type_id::create("reset_sequence");
30         set_type_override_by_type(ALSU_seq_item::get_type(), alsu_seq_item_valid_invalid::get_type());
31
32         if(!uvm_config_db#(virtual ALSU_if)::get (this , "" , "ALSUif" , alsu_config_obj_test.alsu_config_vif ))
33             `uvm_fatal("run_phase" , "test - unable to get the virtual interface ") ;
34
35         uvm_config_db#(ALSU_config)::set (this , "" , "CFG" , alsu_config_obj_test ) ;
36
37     endfunction
38
39     task run_phase(uvm_phase phase) ;
40         super.run_phase(phase);
41         phase.raise_objection(this);
42         `uvm_info("run_phase","reset assert" , UVM_LOW)
43         reset_sequence.start(env.agt.sqr);
44         `uvm_info("run_phase","reset deassert" , UVM_LOW)
45
46         `uvm_info("run_phase","stimulus generation started" , UVM_LOW)
47         main_sequence.start(env.agt.sqr);
48         `uvm_info("run_phase","stimulus generation ended" , UVM_LOW)
49         phase.drop_objection(this);
50     endtask:run_phase
51
52 endclass
53 endpackage

```

```
1 package ALSU_env_pkg ;
2
3 import ALSU_agent_pkg::*;
4 import ALSU_scoreboard_pkg::*;
5 import ALSU_coverage_pkg::*;
6 import uvm_pkg::* ;
7 `include "uvm_macros.svh"
8
9
10 class ALSU_env extends uvm_env;
11     `uvm_component_utils(ALSU_env)
12
13     ALSU_agent agt ;
14     ALSU_scoreboard sb ;
15     ALSU_coverage cov ;
16
17     function new(string name = "ALSU_env" , uvm_component parent = null );
18         super.new(name ,parent ) ;
19     endfunction
20
21     function void build_phase(uvm_phase phase) ;
22         super.build_phase(phase);
23         agt = ALSU_agent::type_id::create("agt",this );
24         sb = ALSU_scoreboard::type_id::create("sb",this );
25         cov = ALSU_coverage::type_id::create("cov",this );
26     endfunction
27
28
29     function void connect_phase(uvm_phase phase) ;
30         agt.agt_ap.connect(sb.sb_export);
31         agt.agt_ap.connect(cov.cov_export);
32     endfunction
33
34 endclass
35 endpackage
```


36- reset sequence

```

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

```

37- Main sequence

```
1 package ALSU_seq_main_pkg;
2
3 import ALSU_seq_item_pkg::*;
4 import uvm_pkg::* ;
5 `include "uvm_macros.svh"
6
7
8 class ALSU_main_sequence extends uvm_sequence #(ALSU_seq_item);
9     `uvm_object_utils(ALSU_main_sequence)
10
11     ALSU_seq_item seq_item ;
12     function new(string name = "ALSU_main_sequence" );
13         super.new(name) ;
14     endfunction
15     task body;
16         repeat(100000)
17             begin
18                 seq_item = ALSU_seq_item::type_id::create("seq_item");
19                 start_item(seq_item);
20                 assert(seq_item.randomize()) ;
21                 finish_item(seq_item);
22             end
23         endtask
24     endclass
25
26 endpackage
```

38- ALSU agent

```
1 package ALSU_agent_pkg ;
2
3 import ALSU_driver_pkg::*;
4 import ALSU_sequencer_pkg::*;
5 import ALSU_monitor_pkg::*;
6 import ALSU_config_pkg::*;
7 import ALSU_seq_item_pkg::* ;
8 import uvm_pkg::* ;
9 `include "uvm_macros.svh"
10
11
12 class ALSU_agent extends uvm_agent;
13     `uvm_component_utils(ALSU_agent)
14
15     ALSU_driver driver ;
16     ALSU_sequencer sqr ;
17     ALSU_monitor mon ;
18     ALSU_config ALSU_cfg ;
19     uvm_analysis_port #(ALSU_seq_item) agt_ap ;
20
21     function new(string name = "ALSU_agent" , uvm_component parent = null );
22         super.new(name ,parent ) ;
23     endfunction
24
```

```

24
25     function void build_phase(uvm_phase phase) ;
26         super.build_phase(phase);
27         if(!uvm_config_db#(ALSU_config)::get (this , "" , "CFG" , ALSU_cfg ) )
28             `uvm_fatal("build_phase" , "driver - unable to get the virtual interface ") ;
29
30         driver = ALSU_driver::type_id::create("driver",this );
31         sqr = ALSU_sequencer::type_id::create("sqr",this );
32         mon = ALSU_monitor::type_id::create("mon",this );
33         agt_ap = new("agt_ap" , this) ;
34     endfunction
35
36     function void connect_phase(uvm_phase phase) ;
37         driver.alsu_driver_vif = ALSU_cfg.alsu_config_vif ;
38         mon.ALSU_vif = ALSU_cfg.alsu_config_vif ;
39         driver.seq_item_port.connect(sqr.seq_item_export) ;
40         mon.mon_ap.connect(agt_ap);
41     endfunction
42 endclass
43 endpackage

```

39- ALSU scoreboard

```

1 package ALSU_scoreboard_pkg ;
2 import ALSU_seq_item_pkg::* ;
3 import uvm_pkg::* ;
4 `include "uvm_macros.svh"
5 class ALSU_scoreboard extends uvm_scoreboard;
6     `uvm_component_utils(ALSU_scoreboard)
7     uvm_analysis_export #(ALSU_seq_item) sb_export ;
8     uvm_tlm_analysis_fifo #(ALSU_seq_item) sb_fifo ;
9     ALSU_seq_item seq_item_cov ;
10    logic [5:0] dataout_ref ;
11    logic [15:0] leds_ref ;
12
13    logic cin_reg, red_op_A_reg, red_op_B_reg, bypass_A_reg, bypass_B_reg, direction_reg, serial_in_reg;
14    logic [2:0] opcode_reg;
15    logic signed [2:0] A_reg, B_reg;
16    logic invalid_red_op, invalid_opcode, invalid;
17    int error_count = 0 ;
18    int correct_count = 0 ;
19    function new(string name = "ALSU_scoreboard" , uvm_component parent = null );
20        super.new(name ,parent ) ;
21    endfunction
22    function void build_phase(uvm_phase phase) ;
23        super.build_phase(phase);
24        sb_export = new("sb_export" , this) ;
25        sb_fifo = new("sb_fifo" , this) ;
26    endfunction
27
28    function void connect_phase(uvm_phase phase) ;
29        sb_export.connect(sb_fifo.analysis_export);
30    endfunction
31

```

```

32 task run_phase(uvm_phase phase) ;
33     super.run_phase(phase);
34     forever
35     begin
36         sb_fifo.get(seq_item_cov) ;
37         ref_model(seq_item_cov) ;
38         if(seq_item_cov.out != dataout_ref && seq_item_cov.leds != leds_ref )
39         begin
40             `uvm_error("run_phase" , $sformatf("comparsion failed trasnsaction received by the dut %s shile \
41             | the reference out %ob" ,seq_item_cov.convert2string , dataout_ref )) ;
42             error_count++ ;
43         end
44         else
45         begin
46             `uvm_info("run_phase" ,seq_item_cov.convert2string_stimulus() , UVM_HIGH ) ;
47             correct_count++ ;
48         end
49     end
50 endtask

```

```

52 task ref_model(ASLU_seq_item seq_item_cov);
53     invalid_red_op = (red_op_A_reg | red_op_B_reg) & (opcode_reg[1] | opcode_reg[2]);
54     invalid_opcode = opcode_reg[1] & opcode_reg[2];
55     invalid = invalid_red_op | invalid_opcode;
56     fork
57     begin
58         if(seq_item_cov.rst) begin
59             cin_reg <= 0;
60             red_op_B_reg <= 0;
61             red_op_A_reg <= 0;
62             bypass_B_reg <= 0;
63             bypass_A_reg <= 0;
64             direction_reg <= 0;
65             serial_in_reg <= 0;
66             opcode_reg <= 0;
67             A_reg <= 0;
68             B_reg <= 0;
69         end else begin
70             cin_reg <= seq_item_cov.cin;
71             red_op_B_reg <= seq_item_cov.red_op_B;
72             red_op_A_reg <= seq_item_cov.red_op_A;
73             bypass_B_reg <= seq_item_cov.bypass_B;
74             bypass_A_reg <= seq_item_cov.bypass_A;
75             direction_reg <= seq_item_cov.direction;
76             serial_in_reg <= seq_item_cov.serial_in;
77             opcode_reg <= seq_item_cov.opcode;
78             A_reg <= seq_item_cov.A;
79             B_reg <= seq_item_cov.B;
80         end
81     end
82

```

```

82
83     begin
84         if(seq_item_cov.rst) begin
85             dataout_ref <= 0;
86         end
87         else begin
88             if (invalid)
89                 dataout_ref <= 0;
90             else if (bypass_A_reg && bypass_B_reg)
91                 dataout_ref <= A_reg;
92             else if (bypass_A_reg)
93                 dataout_ref <= A_reg;
94             else if (bypass_B_reg)
95                 dataout_ref <= B_reg;
96             else begin
97                 case (opcode_reg)
98                     3'h0: begin
99                         if (red_op_A_reg && red_op_B_reg)
100                             dataout_ref <= |A_reg;
101                         else if (red_op_A_reg)
102                             dataout_ref <= |A_reg;
103                         else if (red_op_B_reg)
104                             dataout_ref <= |B_reg;
105                         else
106                             dataout_ref <= A_reg | B_reg;
107                     end
108                     3'h1: begin
109                         if (red_op_A_reg && red_op_B_reg)
110                             dataout_ref <= ^A_reg;
111                             dataout_ref <= ^A_reg;
112                             dataout_ref <= ^A_reg;
113                             else if (red_op_B_reg)
114                                 dataout_ref <= ^B_reg;
115                             else
116                                 dataout_ref <= A_reg ^ B_reg;
117                             end
118                             3'h2:begin
119                                 dataout_ref <= A_reg + B_reg + cin_reg ;
120                                 end
121                             3'h3: dataout_ref <= A_reg * B_reg;
122                             3'h4: begin
123                                 if (direction_reg)
124                                     dataout_ref <= {dataout_ref[4:0], serial_in_reg};
125                                 else
126                                     dataout_ref <= {serial_in_reg, dataout_ref[5:1]};
127                                 end
128                             3'h5: begin
129                                 if (direction_reg)
130                                     dataout_ref <= {dataout_ref[4:0], dataout_ref[5]};
131                                 else
132                                     dataout_ref <= {dataout_ref[0], dataout_ref[5:1]};
133                                 end
134                             default : dataout_ref <= 0 ;
135                             endcase
136                         end
137                     end
138                 end
139             end

```

```

140
141         join
142         if(seq_item_cov.rst) begin
143             leds_ref <= 0;
144         end else begin
145             if (invalid)
146                 leds_ref <= ~leds_ref;
147             else
148                 leds_ref <= 0;
149         end
150     endtask
151
152     function void report_phase(uvm_phase phase) ;
153     super.report_phase(phase);
154     `uvm_info("report_phase", $sformatf("total successful %0d ", correct_count ), UVM_MEDIUM ) ;
155     `uvm_info("report_phase", $sformatf("total FAILED %0d ", error_count ), UVM_MEDIUM ) ;
156 endfunction
157 endclass
158 endpackage

```

10-ALSU coverage

```

1 package ALSU_coverage_pkg ;
2
3 import ALSU_seq_item_pkg::* ;
4 import uvm_pkg::* ;
5 `include "uvm_macros.svh"
6
7
8 class ALSU_coverage extends uvm_component;
9     `uvm_component_utils(ALSU_coverage)
10
11     uvm_analysis_export #(ALSU_seq_item) cov_export ;
12     uvm_tlm_analysis_fifo #(ALSU_seq_item) cov_fifo ;
13     ALSU_seq_item seq_item_cov ;
14     typedef enum logic [2:0] {OR , XOR , ADD , MULT , SHIFT , ROTATE , INVALID_6 , INVALID_7 } reg_e ;
15     typedef enum {Or , Xor , Add , Mult , Shift , Rotate} opcode_valid_e ;
16     localparam MAXPOS = 3 ;
17     localparam MAXNEG = -4 ;
18     localparam zero = 0 ;
19     covergroup cvr_gp ;
20         A_cp : coverpoint seq_item_cov.A {
21             bins A_data_0 = {0} ;
22             bins A_data_max = {MAXPOS} ;
23             bins A_data_min = {MAXNEG} ;
24             bins A_data_default = default ;
25             bins A_data_walkingones[] = {1, 2, -4} iff (seq_item_cov.red_op_A && !seq_item_cov.red_op_B);
26         }
27
28         B_cp : coverpoint seq_item_cov.B {
29             bins B_data_0 = {0} ;
30             bins B_data_max = {MAXPOS} ;
31             bins B_data_min = {MAXNEG} ;
32             bins B_data_default = default ;
33             bins B_data_walkingones[] = {1, 2, -4} iff (!seq_item_cov.red_op_A && seq_item_cov.red_op_B);
34         }

```

```

35
36     ALU_cp : coverpoint seq_item_cov.opcode {
37         bins Bins_shift[] = {SHIFT , ROTATE} ;
38         bins Bins_arith[] = {ADD , MULT} ;
39         bins Bins_bitwise[] = {OR , XOR} ;
40         illegal_bins Bins_invalid = {INVALID_6 , INVALID_7} ;
41     }
42
43     cin_cp : coverpoint seq_item_cov.cin {
44         bins cin_data = {0 , 1} ;
45     }
46
47     direction_cp : coverpoint seq_item_cov.direction {
48         bins direction_data = {0 , 1} ;
49     }
50
51     serial_in_cp : coverpoint seq_item_cov.serial_in {
52         bins serial_in_data = {0 , 1} ;
53     }
54
55     red_op_A_cp : coverpoint seq_item_cov.red_op_A {
56         bins red_op_A_LOW_data = {0} ;
57         bins red_op_A_HIGH_data = {1} ;
58     }
59     red_op_B_cp : coverpoint seq_item_cov.red_op_B {
60         bins red_op_B_LOW_data = {0} ;
61         bins red_op_B_HIGH_data = {1} ;
62     }
63
64
65
66     add_mult_cp1 : cross A_cp , B_cp , ALU_cp
67     {
68
69         bins zero_A_add = binsof(ALU_cp.Bins_arith) && binsof(A_cp.A_data_0) && binsof(B_cp.B_data_0) ;
70         bins max_pos_add = binsof(ALU_cp.Bins_arith) && binsof(A_cp.A_data_max) && binsof(B_cp.B_data_max) ;
71         option.cross_auto_bin_max = 0 ;
72     }
73
74
75
76     opcode_cp2 : cross cin_cp , direction_cp , serial_in_cp , ALU_cp , red_op_A_cp , red_op_B_cp
77     {
78
79         bins cin_add = binsof(cin_cp.cin_data) && binsof(ALU_cp.Bins_arith) intersect {ADD} ;
80         bins serialin_shift = binsof(serial_in_cp.serial_in_data) && binsof(ALU_cp) intersect {SHIFT} ;
81         bins direction_shift_rota = binsof(direction_cp) && binsof(ALU_cp.Bins_shift) ;
82         option.cross_auto_bin_max = 0 ;
83     }
84
85
86
87     or_xor_cp3 : cross A_cp , B_cp , ALU_cp , red_op_A_cp , red_op_B_cp
88     {
89         bins or_xor_data_A = binsof(ALU_cp.Bins_bitwise) && binsof(A_cp.A_data_walkingones) && binsof(B_cp.B_data_0) \
90         && binsof(red_op_B_cp.red_op_B_LOW_data) && binsof(red_op_A_cp.red_op_A_HIGH_data) ;
91         bins or_xor_data_B = binsof(ALU_cp.Bins_bitwise) && binsof(B_cp.B_data_walkingones) && binsof(A_cp.A_data_0) \
92         && binsof(red_op_A_cp.red_op_A_LOW_data) && binsof(red_op_B_cp.red_op_B_HIGH_data) ;
93         option.cross_auto_bin_max = 0 ;
94     }

```

```

96
97     INVALID_cp4 : cross  ALU_cp , red_op_A_cp , red_op_B_cp
98     {
99         bins Bins_shift_data = binsof(ALU_cp.Bins_shift) && ( binsof(red_op_B_cp.red_op_B_HIGH_data) || binsof(red_op_A_cp.red_op_A_HIGH_data) ) ;
100         bins Bins_arith_data = binsof(ALU_cp.Bins_arith) && ( binsof(red_op_B_cp.red_op_B_HIGH_data) || binsof(red_op_A_cp.red_op_A_HIGH_data) ) ;
101         option.cross_auto_bin_max = 0 ;
102     }
103
104 endgroup
105 function new(string name = "ALSU_sequencer" , uvm_component parent = null );
106     super.new(name ,parent ) ;
107     cvr_gp=new();
108 endfunction
109
110 function void build_phase(uvm_phase phase) ;
111     super.build_phase(phase);
112     cov_export = new("cov_export" , this) ;
113     cov_fifo = new("cov_fifo" , this) ;
114
115 endfunction
116
117 function void connect_phase(uvm_phase phase) ;
118     super.connect_phase(phase);
119     cov_export.connect(cov_fifo.analysis_export);
120 endfunction
121
122 task run_phase(uvm_phase phase) ;
123     super.run_phase(phase);
124     forever
125     begin
126         cov_fifo.get(seq_item_cov) ;
127         cvr_gp.sample();
128     end
129 endtask
130 endclass
131 endpackage

```

11- ALSU sequencer

```

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

```


12-ALSU monitor

```
1  package ALSU_monitor_pkg ;
2
3      import uvm_pkg::* ;
4      import ALSU_seq_item_pkg::* ;
5      `include "uvm_macros.svh"
6
7      class ALSU_monitor extends uvm_monitor ;
8          `uvm_component_utils(ALSU_monitor)
9
10         virtual ALSU_if ALSU_vif ;
11         ALSU_seq_item seq_item ;
12
13         uvm_analysis_port #(ALSU_seq_item) mon_ap ;
14
15         function new(string name = "ALSU_monitor" , 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             mon_ap = new("mon_ap" , this) ;
22         endfunction
23
24         task run_phase(uvm_phase phase) ;
25             super.run_phase(phase);
26             forever
27                 begin
28                     seq_item = ALSU_seq_item::type_id::create("seq_item");
29
30                     @(negedge ALSU_vif.clk ) ;
31                     seq_item.serial_in = ALSU_vif.serial_in ;
32                     seq_item.red_op_A = ALSU_vif.red_op_A ;
33                     seq_item.red_op_B = ALSU_vif.red_op_B ;
34                     seq_item.bypass_A = ALSU_vif.bypass_A ;
35                     seq_item.bypass_B = ALSU_vif.bypass_B ;
36                     seq_item.direction = ALSU_vif.direction ;
37                     seq_item.serial_in = ALSU_vif.serial_in ;
38                     seq_item.opcode = ALSU_vif.opcode ;
39                     seq_item.A = ALSU_vif.A ;
40                     seq_item.B = ALSU_vif.B ;
41                     seq_item.cin = ALSU_vif.cin ;
42                     mon_ap.write(seq_item);
43                     `uvm_info("run_phase" ,seq_item.convert2string_stimulus() , UVM_HIGH )
44
45                 end
46             endtask
47         endclass
48     endpackage
49
```

13-alsu_driver

```

1  package ALSU_driver_pkg ;
2
3      import uvm_pkg::* ;
4      import ALSU_seq_item_pkg::* ;
5      `include "uvm_macros.svh"
6
7
8      class ALSU_driver extends uvm_driver #(ALSU_seq_item);
9          `uvm_component_utils(ALSU_driver)
10         virtual ALSU_if alsu_driver_vif ;
11         ALSU_seq_item seq_item ;
12
13         function new(string name = "ALSU_driver" , uvm_component parent = null );
14             super.new(name ,parent ) ;
15         endfunction
16
17         task run_phase(uvm_phase phase) ;
18             super.run_phase(phase);
19             alsu_driver_vif.rst = 1 ;
20             alsu_driver_vif.red_op_A = 0 ;
21             alsu_driver_vif.red_op_B = 0 ;
22             alsu_driver_vif.bypass_A = 0 ;
23             alsu_driver_vif.bypass_B = 0 ;
24             alsu_driver_vif.direction = 0 ;
25             alsu_driver_vif.serial_in = 0 ;
26             alsu_driver_vif.opcode = 0 ;
27             alsu_driver_vif.A = 0 ;
28             alsu_driver_vif.B = 0 ;
29             alsu_driver_vif.cin = 0 ;
30
31             @(negedge alsu_driver_vif.clk ) ;
32             alsu_driver_vif.rst = 0 ;
33
34             forever
35             begin
36                 seq_item = ALSU_seq_item::type_id::create("seq_item");
37                 seq_item_port.get_next_item(seq_item);
38                 @(negedge alsu_driver_vif.clk ) ;
39                 alsu_driver_vif.serial_in = seq_item.serial_in ;
40                 alsu_driver_vif.red_op_A = seq_item.red_op_A ;
41                 alsu_driver_vif.red_op_B = seq_item.red_op_B ;
42                 alsu_driver_vif.bypass_A = seq_item.bypass_A ;
43                 alsu_driver_vif.bypass_B = seq_item.bypass_B ;
44                 alsu_driver_vif.direction = seq_item.direction ;
45                 alsu_driver_vif.serial_in = seq_item.serial_in ;
46                 alsu_driver_vif.opcode = seq_item.opcode ;
47                 alsu_driver_vif.A = seq_item.A ;
48                 alsu_driver_vif.B = seq_item.B ;
49                 alsu_driver_vif.cin = seq_item.cin ;
50                 seq_item_port.item_done();
51                 `uvm_info("run_phase" ,seq_item.convert2string_stimulus() , UVM_HIGH )
52             end
53         endtask
54     endclass
55 endpackage

```

14-ALSU seq_item

```
1 package ALSU_seq_item_pkg ;
2 import uvm_pkg::* ;
3 `include "uvm_macros.svh"
4
5 class ALSU_seq_item extends uvm_sequence_item;
6   `uvm_object_utils(ALSU_seq_item)
7   typedef enum logic [2:0] {OR , XOR , ADD , MULT , SHIFT , ROTATE , INVALID_6 , INVALID_7 } reg_e ;
8   typedef enum {Or , Xor , Add , Mult , Shift , Rotate} opcode_valid_e ;
9   localparam MAXPOS = 3 ;
10  localparam MAXNEG = -4 ;
11  localparam zero = 0 ;
12  rand bit rst, cin, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
13  rand bit [2:0] opcode;
14  rand bit signed [2:0] A, B;
15  logic [15:0] leds;
16  logic signed [5:0] out;
17
18  constraint rst_n {rst dist {0:/99 , 1:/1 } ; }
19
20  constraint input_A {
21    if ( (opcode == ADD ) || (opcode == MULT ) )
22    {
23      A dist { MAXPOS:/25 , MAXNEG:/25 , zero:/25 , {3'b010 , 3'b001 , 3'b111 , 3'b110 , 3'b101}:/25 } ;
24    }
25    else if (((opcode == XOR ) || (opcode == OR )) && (red_op_A == 1) && (red_op_B == 0) )
26    {
27      B == 0 ;
28      A dist {3'b001:/30 , 3'b010:/30 , 3'b100:/30 , 3'b000:/5 , 3'b011:/5 , 3'b101:/5 , 3'b110:/5 , 3'b111:/5 } ;
29    }
30    else
31    {
32      A inside { [MAXNEG : MAXPOS] } ;
33    }
34  }
35
36
37  constraint input_B {
38    if ( (opcode == ADD ) || (opcode == MULT ) )
39    {
40      B dist { MAXPOS:/25 , MAXNEG:/25 , zero:/25 , {3'b010 , 3'b001 , 3'b111 , 3'b110 , 3'b101}:/25 } ;
41    }
42    else if (((opcode == XOR ) || (opcode == OR )) && (red_op_B == 1) && (red_op_A == 0) )
43    {
44      A == 0 ;
45      B dist {3'b001:/30 , 3'b010:/30 , 3'b100:/30 , 3'b000:/5 , 3'b011:/5 , 3'b101:/5 , 3'b110:/5 , 3'b111:/5 } ;
46    }
47    else
48    {
49      B inside { [MAXNEG : MAXPOS] } ;
50    }
51  }
52
53  constraint input_opcode {opcode dist {[0:3]:/50 , [4:5]:/50 } ; }
54
55  constraint input_bypass_A {bypass_A dist {1:=90 , 0:=10 } ; }
56  constraint input_bypass_B {bypass_B dist {1:=90 , 0:=10 } ; }
57
58  constraint input_red_op_A {red_op_A dist {1:/10 , 0:/90 } ; }
59  constraint input_red_op_B {red_op_B dist {1:/10 , 0:/90 } ; }
60
61  function new(string name = "ALSU_seq_item" );
62    super.new(name ) ;
63  endfunction
64
65  function string convert2string();
66  return $sformatf ("%s , rst = %0d , serial_in = %0d , direction = %0d , cin = %0d , red_op_A = %0d , red_op_B = %0d , bypass_A = %0d \
67  , bypass_B = %0d , opcode = %0d , A = %0d , B = %0d , leds = %0d , out = %0d" , super.convert2string , rst , serial_in , \
68  direction , cin , red_op_A , red_op_B , bypass_A , bypass_B , opcode , A , B , leds , out ) ;
69  endfunction
70
71  function string convert2string_stimulus();
72  return $sformatf ("rst = %0d , serial_in = %0d , direction = %0d , cin = %0d , red_op_A = %0d , red_op_B = %0d , bypass_A = %0d \
73  , bypass_B = %0d , opcode = %0d , A = %0d , B = %0d " , rst , serial_in , direction , cin , red_op_A , red_op_B \
74  , bypass_A , bypass_B , opcode , A , B ) ;
75  endfunction
76
77  endclass
78 endpackage
```

15-ALSU seq_item_valid_invalid

```
1 package alsu_seq_item_valid_invalid_pkg ;
2 import uvm_pkg::* ;
3 import ALSU_seq_item_pkg::* ;
4 `include "uvm_macros.svh"
5
6 class alsu_seq_item_valid_invalid extends ALSU_seq_item;
7     `uvm_object_utils(alsu_seq_item_valid_invalid)
8
9     constraint input_opcode {opcode dist {[0:3]:/40 , [4:5]:/40 , [6:7]:/20 } ; }
10
11     constraint input_bypass_A {bypass_A dist {1:=90 , 0:=10 } ; }
12     constraint input_bypass_B {bypass_B dist {1:=90 , 0:=10 } ; }
13
14     constraint input_red_op_A {red_op_A dist {1:/10 , 0:/90 } ; }
15     constraint input_red_op_B {red_op_B dist {1:/10 , 0:/90 } ; }
16
17
18     function new(string name = "alsu_seq_item_valid_invalid" );
19         super.new(name) ;
20     endfunction
21
22     function string convert2string();
23     return $sformatf ("rst = %0d , serial_in = %0d , direction = %0d , cin = %0d , red_op_A = %0d , red_op_B = %0d , bypass_A = %0d , bypass_B = %0d \
24 , opcode = %0d , A = %0d , B = %0d , leds = %0d , out = %0d" , super.convert2string , rst , serial_in , direction , cin , red_op_A , red_op_B \
25 , bypass_A , bypass_B , opcode , A , B , leds , out ) ;
26     endfunction
27
28     function string convert2string_stimulus();
29     return $sformatf ("rst = %0d , serial_in = %0d , direction = %0d , cin = %0d , red_op_A = %0d , red_op_B = %0d , bypass_A = %0d , bypass_B = %0d \
30 , opcode = %0d , A = %0d , B = %0d " , rst , serial_in , direction , cin , red_op_A , red_op_B , bypass_A , bypass_B , opcode , A , B ) ;
31     endfunction
32
33 endclass
34 endpackage
```

16-alsu_config_obj

```
1 package ALSU_config_pkg ;
2
3     import uvm_pkg::* ;
4     `include "uvm_macros.svh"
5
6     class ALSU_config extends uvm_object;
7         `uvm_object_utils(ALSU_config)
8
9         virtual ALSU_if alsu_config_vif ;
10         function new(string name = "ALSU_config");
11             super.new(name) ;
12         endfunction
13     endclass
14 endpackage
```

17- Do file

```
C: > Users > CS > Downloads > Kareem Wasem Diploma > session6_Assignmen > ALSU > run.do
1  vlib work
2  vlog *v +cover
3  vsim -voptargs=+acc work.ALSU_top -classdebug -uvmmcontrol=all -cover
4  add wave /ALSU_top/ALSUif/*
5  coverage save top.ucdb -onexit
6  run -all
7  quit -sim
8  vcover report top.ucdb -all -details -output coverage.txt
```

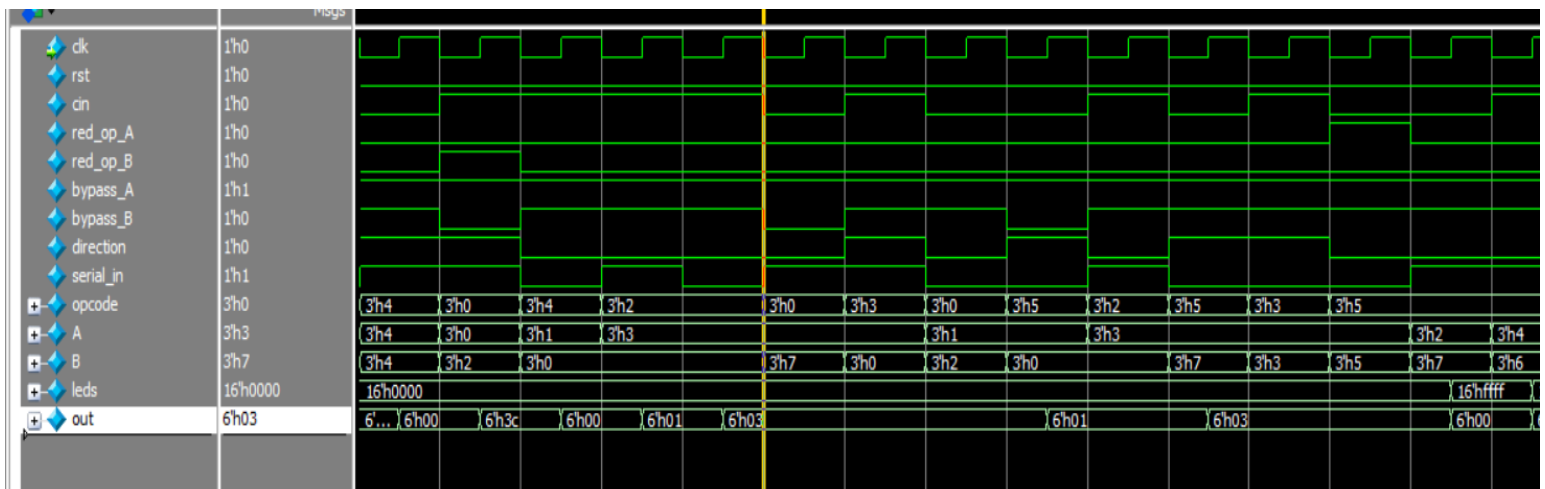
18-Transcript without the override method call

```
# * To turn off, set 'recording_detail' to off: *
# * uvm_config_db#(int) ::set(null, "", "recording_detail", 0); *
# * uvm_config_db#(uvm_bitstream_t)::set(null, "", "recording_detail", 0); *
# *****
# UVM_INFO ALSU_test.sv(43) @ 4: uvm_test_top [run_phase] reset deassert
# UVM_INFO ALSU_test.sv(45) @ 4: uvm_test_top [run_phase] stimulus generation started
# UVM_INFO ALSU_test.sv(47) @ 200004: uvm_test_top [run_phase] stimulus generation ended
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1268) @ 200004: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO ALSU_scoreboard.sv(160) @ 200004: uvm_test_top.env.sb [report_phase] total successful 100002
# UVM_INFO ALSU_scoreboard.sv(161) @ 200004: uvm_test_top.env.sb [report_phase] total FAILED 0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO : 10
# 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] 4
#
# ** Note: $finish : C:/questasim64_10.4c/win64/./verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
# Time: 200004 ns Iteration: 61 Instance: /ALSU_top
```

19-Transcript without the override method call

```
# ** Error: (vsim-8565) Illegal state bin was hit at value=6. The bin counter for the illegal bin '\ALSU_coverage_pkg::ALSU_coverage::cvr_gp .ALU_cp.Bins_invalid' is 19899.
# Time: 199936 ns Iteration: 2 Region: /uvm_pkg::uvm_task_phase::execute
# ** Error: (vsim-8565) Illegal state bin was hit at value=6. The bin counter for the illegal bin '\ALSU_coverage_pkg::ALSU_coverage::cvr_gp .ALU_cp.Bins_invalid' is 19900.
# Time: 199938 ns Iteration: 2 Region: /uvm_pkg::uvm_task_phase::execute
# ** Error: (vsim-8565) Illegal state bin was hit at value=7. The bin counter for the illegal bin '\ALSU_coverage_pkg::ALSU_coverage::cvr_gp .ALU_cp.Bins_invalid' is 19901.
# Time: 199970 ns Iteration: 2 Region: /uvm_pkg::uvm_task_phase::execute
# ** Error: (vsim-8565) Illegal state bin was hit at value=6. The bin counter for the illegal bin '\ALSU_coverage_pkg::ALSU_coverage::cvr_gp .ALU_cp.Bins_invalid' is 19902.
# Time: 199986 ns Iteration: 2 Region: /uvm_pkg::uvm_task_phase::execute
# ** Error: (vsim-8565) Illegal state bin was hit at value=6. The bin counter for the illegal bin '\ALSU_coverage_pkg::ALSU_coverage::cvr_gp .ALU_cp.Bins_invalid' is 19903.
# Time: 200004 ns Iteration: 2 Region: /uvm_pkg::uvm_task_phase::execute
# UVM_INFO ALSU_test.sv(47) @ 200004: uvm_test_top [run_phase] stimulus generation ended
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1268) @ 200004: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO ALSU_scoreboard.sv(160) @ 200004: uvm_test_top.env.sb [report_phase] total successful 100002
# UVM_INFO ALSU_scoreboard.sv(161) @ 200004: uvm_test_top.env.sb [report_phase] total FAILED 0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO : 10
# 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] 4
```





















20- Waveform



21-Assertion

[illegible]

22-Assertion coverage

Name	Language	Enabled	Log	Count	At least	Limit	Weight	Cmpit %	Cmpit graph	Included	Memory	Peak memory	Peak memory time	Cumulative reads
▲ /ALSU_top/dut/sva... SVA	✓	Off	15	1	Unli...	1	100%		✓		0	0	0 ns	0
▲ /ALSU_top/dut/sva... SVA	✓	Off	14	1	Unli...	1	100%		✓		0	0	0 ns	0
▲ /ALSU_top/dut/sva... SVA	✓	Off	74	1	Unli...	1	100%		✓		0	0	0 ns	0
▲ /ALSU_top/dut/sva... SVA	✓	Off	8	1	Unli...	1	100%		✓		0	0	0 ns	0
▲ /ALSU_top/dut/sva... SVA	✓	Off	15	1	Unli...	1	100%		✓		0	0	0 ns	0
▲ /ALSU_top/dut/sva... SVA	✓	Off	96	1	Unli...	1	100%		✓		0	0	0 ns	0
▲ /ALSU_top/dut/sva... SVA	✓	Off	89	1	Unli...	1	100%		✓		0	0	0 ns	0
▲ /ALSU_top/dut/sva... SVA	✓	Off	105	1	Unli...	1	100%		✓		0	0	0 ns	0
▲ /ALSU_top/dut/sva... SVA	✓	Off	114	1	Unli...	1	100%		✓		0	0	0 ns	0
▲ /ALSU_top/dut/sva... SVA	✓	Off	94	1	Unli...	1	100%		✓		0	0	0 ns	0
▲ /ALSU_top/dut/sva... SVA	✓	Off	89	1	Unli...	1	100%		✓		0	0	0 ns	0
▲ /ALSU_top/dut/sva... SVA	✓	Off	111	1	Unli...	1	100%		✓		0	0	0 ns	0
▲ /ALSU_top/dut/sva... SVA	✓	Off	512	1	Unli...	1	100%		✓		0	0	0 ns	0
▲ /ALSU_top/dut/sva... SVA	✓	Off	512	1	Unli...	1	100%		✓		0	0	0 ns	0
▲ /ALSU_top/dut/sva... SVA	✓	Off	18	1	Unli...	1	100%		✓		0	0	0 ns	0
▲ /ALSU_top/dut/sva... SVA	✓	Off	14	1	Unli...	1	100%		✓		0	0	0 ns	0
▲ /ALSU_top/dut/sva... SVA	✓	Off	25	1	Unli...	1	100%		✓		0	0	0 ns	0
▲ /ALSU_top/dut/sva... SVA	✓	Off	20	1	Unli...	1	100%		✓		0	0	0 ns	0
▲ /ALSU_top/dut/sva... SVA	✓	Off	17079	1	Unli...	1	100%		✓		0	0	0 ns	0
▲ /ALSU_top/dut/sva... SVA	✓	Off	1627	1	Unli...	1	100%		✓		0	0	0 ns	0