

UVM
Assignment6

By: Mohamed Sayed Mohamed Soliman

1- 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

```

2- 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
```

3- 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
```

4- 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 uvm_pkg::* ;
8 `include "uvm_macros.svh"
9
10
11 class ALSU_test extends uvm_test;
12     `uvm_component_utils(ALSU_test)
13     ALSU_env env ;
14     ALSU_config alsu_config_obj_test ;
15     ALSU_reset_sequence reset_sequence ;
16     ALSU_main_sequence main_sequence ;
17
18     function new(string name = "ALSU_test" , uvm_component parent = null );
19         super.new(name ,parent ) ;
20     endfunction
21
22     function void build_phase(uvm_phase phase) ;
23         super.build_phase(phase);
24         env = ALSU_env::type_id::create("env",this );
25         alsu config obj test = ALSU config::type_id::create("alsu config obj test");
26         main_sequence = ALSU_main_sequence::type_id::create("main_sequence" );
27         reset_sequence = ALSU_reset_sequence::type_id::create("reset_sequence");
28         if(!uvm_config_db#(virtual ALSU_if)::get (this , "" , "ALSUif" , alsu_config_obj_test.alsu_config_vif ))
29             `uvm_fatal("run_phase" , "test - unable to get the virtual interface ") ;
30
31         uvm_config_db#(ALSU_config)::set (this , "" , "CFG" , alsu_config_obj_test ) ;
32
33     endfunction
34
35     task run_phase(uvm_phase phase) ;
36         super.run_phase(phase);
37         phase.raise_objection(this);
38         `uvm_info("run_phase","reset assert" , UVM_LOW)
39         reset_sequence.start(env.agt.sqr);
40         `uvm_info("run_phase","reset deassert" , UVM_LOW)
41
42         `uvm_info("run_phase","stimulus generation started" , UVM_LOW)
43         main_sequence.start(env.agt.sqr);
44         `uvm_info("run_phase","stimulus generation ended" , UVM_LOW)
45         phase.drop_objection(this);
46     endtask:run_phase
47
48 endclass
49 endpackage
```

5- 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
```

6- 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
```

7- 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
```

8- 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

```

9- 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
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  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
```

15-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
```

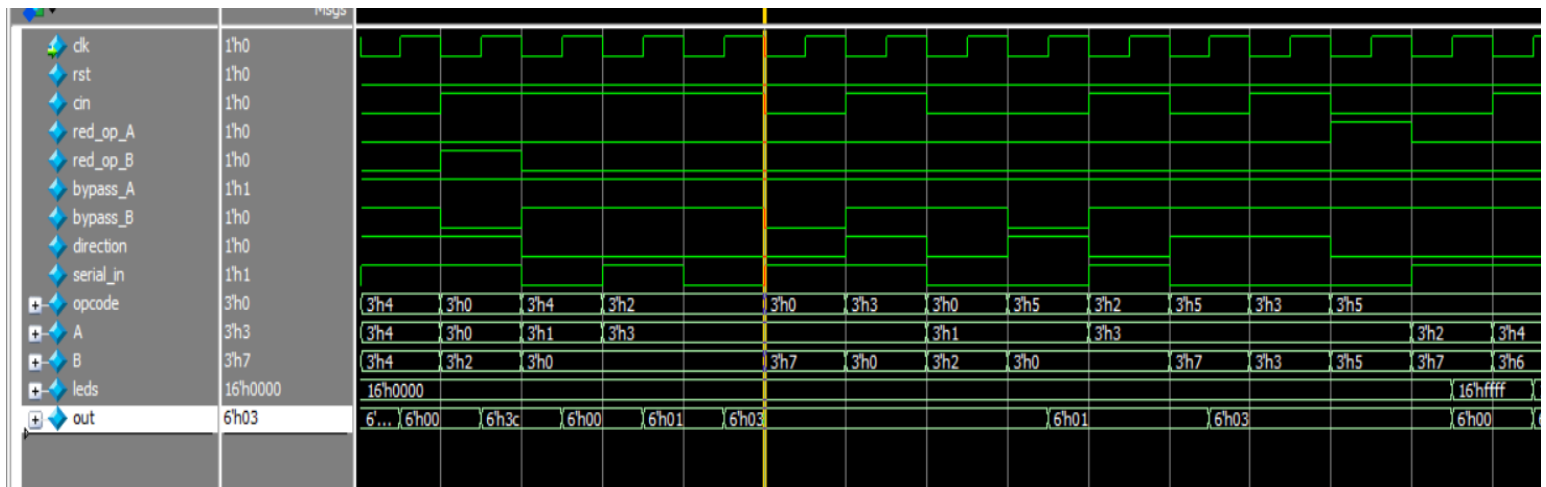
16- Do file

```
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 -uvmcontrol=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
```

17-Transcript

```
# TIME: 199/04 ns iteration: 4 REGION: /uvm_pkg::uvm_test_phase::execute
# UVM_INFO ALSU_test.sv(44) @ 200004: uvm_test_top [run_phase] stimulus generation ended
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_object.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
# [RNIST] 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)
```

18- Waveform



19-Assertion

+	/ALSU_top/dut/sva...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge ALSUif.clk) disa...	✓
+	/ALSU_top/dut/sva...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge ALSUif.clk) disa...	✓
+	/ALSU_top/dut/sva...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge ALSUif.clk) disa...	✓
+	/ALSU_top/dut/sva...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge ALSUif.clk) disa...	✓
+	/ALSU_top/dut/sva...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge ALSUif.clk) disa...	✓
+	/ALSU_top/dut/sva...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge ALSUif.clk) disa...	✓
+	/ALSU_top/dut/sva...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge ALSUif.clk) disa...	✓
+	/ALSU_top/dut/sva...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge ALSUif.clk) disa...	✓
+	/ALSU_top/dut/sva...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge ALSUif.clk) disa...	✓
+	/ALSU_top/dut/sva...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge ALSUif.clk) disa...	✓
+	/ALSU_top/dut/sva...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge ALSUif.clk) disa...	✓
+	/ALSU_top/dut/sva...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge ALSUif.clk) disa...	✓
+	/ALSU_top/dut/sva...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge ALSUif.clk) disa...	✓
+	/ALSU_top/dut/sva...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge ALSUif.clk) disa...	✓
+	/ALSU_top/dut/sva...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge ALSUif.clk) disa...	✓
+	/ALSU_top/dut/sva...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge ALSUif.clk) disa...	✓
+	/ALSU_top/dut/sva...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge ALSUif.clk) disa...	✓

20-Assertion coverage

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

21-Code Coverage

```
1 Coverage Report by file with details
2
3 =====
4 == File: ALSU.v
5 =====
6 Statement Coverage:
7   Enabled Coverage      Active      Hits      Misses % Covered
8   -----
9   Stmts                48          48          0    100.0
10
11 =====Statement Details=====
12
13 Statement Coverage for file ALSU.v --
14
15 1 //import pack_ALSU::*;
16 2
17 3 module ALSU(A, B, cin, serial_in, red_op_A, red_op_B, opcode, bypass_A, bypass_B, clk, rst, direction, leds, out);
18 4 parameter INPUT_PRIORITY = "A";
19 5 parameter FULL_ADDER = "ON";
20 6 input clk, rst, cin, red_op_A, red_op_B, bypass_A, bypass_B, direction, serial_in;
21 7 input [2:0] opcode;
22 8 input signed [2:0] A, B; // first bug [ ] -> we must put it signed
23 9 output reg [15:0] leds;
24 10 output reg signed [5:0] out; // second bug [ ] -> we must put it signed
25 11
26 12 reg cin_reg, red_op_A_reg, red_op_B_reg, bypass_A_reg, bypass_B_reg, direction_reg, serial_in_reg;
27 13 reg [2:0] opcode_reg;
28 14 reg signed [2:0] A_reg, B_reg;
29 15 wire invalid_red_op, invalid_opcode, invalid;
30 16
31 17 //Invalid handling
32 18 1 87212 assign invalid_red_op = (red_op_A_reg | red_op_B_reg) & (opcode_reg[1] | opcode_reg[2]);
33 19 1 81013 assign invalid_opcode = opcode_reg[1] & opcode_reg[2];
34 20 1 26404 assign invalid = invalid_red_op | invalid_opcode;
35 21
```

```
137
138 Branch Coverage:
139   Enabled Coverage      Active      Hits      Misses % Covered
140   -----
141   Branches                29          29          0    100.0
142
143 =====Branch Details=====
144
145 Branch Coverage for file ALSU.v --
146
147 -----IF Branch-----
148 24 100002 Count coming in to IF
149 24 1 2 if(rst) begin
150 35 1 100000 end else begin
151 Branch totals: 2 hits of 2 branches = 100.0%
152
153 -----IF Branch-----
154 51 100003 Count coming in to IF
155 51 1 2 if(rst) begin
156 54 1 15395 if (invalid)
157 56 1 84606 else
158 Branch totals: 3 hits of 3 branches = 100.0%
159
160 -----IF Branch-----
161 63 99987 Count coming in to IF
162 63 1 2 if(rst) begin
163 67 1 15394 if (invalid)
164 69 1 68626 else if (bypass_A_reg && bypass_B_reg)
165 71 1 7600 else if (bypass_A_reg)
166 73 1 7542 else if (bypass_B_reg)
167 75 1 823 else begin
168 Branch totals: 6 hits of 6 branches = 100.0%
169
```

384	10	out[4]	1	1	100.00
385	10	out[3]	1	1	100.00
386	10	out[2]	1	1	100.00
387	10	out[1]	1	1	100.00
388	10	out[0]	1	1	100.00
389	12	serial_in_reg	1	1	100.00
390	12	red_op_B_reg	1	1	100.00
391	12	red_op_A_reg	1	1	100.00
392	12	direction_reg	1	1	100.00
393	12	cin_reg	1	1	100.00
394	12	bypass_B_reg	1	1	100.00
395	12	bypass_A_reg	1	1	100.00
396	13	opcode_reg[2]	1	1	100.00
397	13	opcode_reg[1]	1	1	100.00
398	13	opcode_reg[0]	1	1	100.00
399	14	B_reg[2]	1	1	100.00
400	14	B_reg[1]	1	1	100.00
401	14	B_reg[0]	1	1	100.00
402	14	A_reg[2]	1	1	100.00
403	14	A_reg[1]	1	1	100.00
404	14	A_reg[0]	1	1	100.00
405	15	invalid_red_op	1	1	100.00
406	15	invalid_opcode	1	1	100.00
407	15	invalid	1	1	100.00

409	Total Node Count	=	58
410	Toggled Node Count	=	58
411	Untoggled Node Count	=	0
412			
413	Toggle Coverage	=	100.0% (116 of 116 bins)

2374	TYPE	/ALSU_coverage_pkg/ALSU_coverage/cvr_gp	100.0%	100	Covered
2375	covered/total bins:		34	34	
2376	missing/total bins:		0	34	
2377	% Hit:		100.0%	100	
2378	Coverpoint cvr_gp::A_cp		100.0%	100	Covered
2379	covered/total bins:		6	6	
2380	missing/total bins:		0	6	
2381	% Hit:		100.0%	100	
2382	bin A_data_0	22895		1	Covered
2383	bin A_data_max	20923		1	Covered
2384	bin A_data_min	9695		1	Covered
2385	bin A_data_walkingones[-4]	1090		1	Covered
2386	bin A_data_walkingones[1]	1108		1	Covered
2387	bin A_data_walkingones[2]	1169		1	Covered
2388	default bin A_data_default	27459			Occurred
2389	Coverpoint cvr_gp::B_cp		100.0%	100	Covered
2390	covered/total bins:		6	6	
2391	missing/total bins:		0	6	
2392	% Hit:		100.0%	100	
2393	bin B_data_0	23008		1	Covered
2394	bin B_data_max	20917		1	Covered
2395	bin B_data_min	9490		1	Covered
2396	bin B_data_walkingones[-4]	1137		1	Covered
2397	bin B_data_walkingones[1]	1193		1	Covered
2398	bin B_data_walkingones[2]	1169		1	Covered
2399	default bin B_data_default	27339			Occurred
2400	Coverpoint cvr_gp::ALU_cp		100.0%	100	Covered
2401	covered/total bins:		6	6	
2402	missing/total bins:		0	6	
2403	% Hit:		100.0%	100	
2404	illegal_bin Bins_invalid	1024			Occurred
2405	bin Bins_shift[4]	26051		1	Covered
2406	bin Bins_shift[5]	25890		1	Covered
2407	bin Bins_arith[2]	11715		1	Covered
2408					

2400 DIRECTIVE COVERAGE.

Name	Design	Design	Lang	File(Line)	Count	Status
	Unit	UnitType				
/ALSU_top/dut/sva/dollar1_cover	SVA	Verilog	SVA	sva_t.sv(146)	15	Covered
/ALSU_top/dut/sva/dollar2_cover	SVA	Verilog	SVA	sva_t.sv(147)	14	Covered
/ALSU_top/dut/sva/dollar3_cover	SVA	Verilog	SVA	sva_t.sv(148)	74	Covered
/ALSU_top/dut/sva/dollar4_cover	SVA	Verilog	SVA	sva_t.sv(149)	8	Covered
/ALSU_top/dut/sva/dollar5_cover	SVA	Verilog	SVA	sva_t.sv(150)	15	Covered
/ALSU_top/dut/sva/dollar6_cover	SVA	Verilog	SVA	sva_t.sv(151)	96	Covered
/ALSU_top/dut/sva/dollar7_cover	SVA	Verilog	SVA	sva_t.sv(152)	89	Covered
/ALSU_top/dut/sva/dollar8_cover	SVA	Verilog	SVA	sva_t.sv(153)	105	Covered
/ALSU_top/dut/sva/dollar9_cover	SVA	Verilog	SVA	sva_t.sv(154)	114	Covered
/ALSU_top/dut/sva/dollar10_cover	SVA	Verilog	SVA	sva_t.sv(155)	94	Covered
/ALSU_top/dut/sva/dollar11_cover	SVA	Verilog	SVA	sva_t.sv(156)	89	Covered
/ALSU_top/dut/sva/dollar12_cover	SVA	Verilog	SVA	sva_t.sv(157)	111	Covered
/ALSU_top/dut/sva/dollar13_cover	SVA	Verilog	SVA	sva_t.sv(158)	512	Covered
/ALSU_top/dut/sva/dollar14_cover	SVA	Verilog	SVA	sva_t.sv(159)	512	Covered
/ALSU_top/dut/sva/dollar15_cover	SVA	Verilog	SVA	sva_t.sv(160)	18	Covered
/ALSU_top/dut/sva/dollar16_cover	SVA	Verilog	SVA	sva_t.sv(161)	14	Covered
/ALSU_top/dut/sva/dollar17_cover	SVA	Verilog	SVA	sva_t.sv(162)	25	Covered
/ALSU_top/dut/sva/dollar18_cover	SVA	Verilog	SVA	sva_t.sv(163)	20	Covered
/ALSU_top/dut/sva/dollar19_cover	SVA	Verilog	SVA	sva_t.sv(164)	17079	Covered
/ALSU_top/dut/sva/dollar20_cover	SVA	Verilog	SVA	sva_t.sv(165)	1627	Covered
TOTAL DIRECTIVE COVERAGE: 100.0% COVERS: 20						