

# AXI4 Verification Project

<b>Eslam fathy</b>
<b>Abdelrahman Ahmed</b>

# I. Memory


## 1. Interface

```
1 interface memory_interface (input bit ACLK);
2
3     //memory signals
4
5     logic      ARESETn;
6     logic      mem_en;
7     logic      mem_we;
8     logic [9:0] mem_addr;
9     logic [31:0] mem_wdata;
10    logic [31:0] mem_rdata;
11
12    modport dut (
13        input  ACLK,
14        input  ARESETn,
15        input  mem_en,
16        input  mem_we,
17        input  mem_addr,
18        input  mem_wdata,
19        output mem_rdata
20    );
21
22    modport tb (
23        input  ACLK,
24        output ARESETn,
25        output mem_en,
26        output mem_we,
27        output mem_addr,
28        output mem_wdata,
29        input  mem_rdata
30    );
31
32
33 endinterface
```

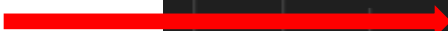
```
1 `include "memory_interface.sv"
2 `include "mem_test.sv"
3 `include "axi_memory.sv"
4 module top();
5
6     //memory signals
7
8     logic      mem_en;
9     logic      mem_we;
10    logic [9:0] mem_addr;
11    logic [31:0] mem_wdata;
12    logic [31:0] mem_rdata;
13
14    logic ACLK;
15    initial begin
16        ACLK = 0;
17    forever begin
18        #5ns ACLK = ~ACLK;
19    end
20 end
21
22    memory_interface inter (ACLK);
23    axi4_memory dut (inter.dut);
24    mem_test tb (inter.tb);
25    mem_assert check (inter.dut);
26
27 endmodule
28
```

## 2. Design bug

```
// Memory write
always @(posedge clk) begin
    if (!rst_n)
        mem_rdata <= 0;
    else if (mem_en) begin
        if (mem_we)
            memory[mem_addr] <= mem_wdata;
        else
            mem_rdata <= (mem_addr % 'hF == 0)? memory[mem_addr-1] & 'hFFF_FFFF : memory[mem_addr] ;
    end
end
```



```
// Memory write and read logic
always @(posedge inter.ACLK) begin
    if (!inter.ARESETn)
        inter.mem_rdata <= 0;
    else if (inter.mem_en) begin
        if (inter.mem_we)
            memory[inter.mem_addr] <= inter.mem_wdata;
        else
            inter.mem_rdata <= memory[inter.mem_addr];
    end
end
```



## 3. Class

```
1 class memory_class;
2     // Memory signals
3     rand logic      mem_en;
4     rand logic      mem_we;
5     rand logic [9:0] mem_addr;
6     rand logic [31:0] mem_wdata;
7
8     // Constraint for memory enable
9     constraint mem_en_const {
10         mem_en dist {0 := 1, 1 := 9};
11     }
12
13     // Constraint for memory write enable
14     constraint mem_we_const {
15         if (mem_en)
16             mem_we dist {0 := 6, 1 := 4}; // 0 = read, 1 = write
17         else
18             mem_we == 0;
19     }
20
21     // Address distribution constraint
22     constraint addr_c {
23         mem_addr inside {[10'd0 : 10'd1023]};
24     };
25
26     // Address corner case constraint
27     constraint addr_corner_c {
28         mem_addr inside {10'd0, 10'd1, 10'd7, 10'd15, 10'd31, 10'd63, 10'd127, 10'd255, 10'd511,
29             10'd1023, 10'b1010101010, 10'b0101010101, 10'b1111111111, 10'b1111100000,
30             10'b0000011111};
31     }
32
```

```

34     constraint data_c1 {
35         mem_wdata inside {
36             [32'd0           : 32'd255],
37             [32'd256        : 32'd1023],
38             [32'd1024       : 32'd4095],
39             [32'd4096       : 32'd16383],
40             [32'd16384      : 32'd32767]
41         };
42     }
43
44     constraint data_c2 {
45         mem_wdata inside {
46             [32'b0           : 32'b1],
47             [32'd32768       : 32'd49151],
48             [32'd49152      : 32'd65535],
49             [32'd65536      : 32'd262143],
50             [32'd262144     : 32'd1048575]
51         };
52     }
53
54     constraint data_c3 {
55         mem_wdata inside {
56             [32'b0           : 32'b1],
57             [32'd1048576     : 32'd16777215],
58             [32'd16777216   : 32'd268435455],
59             [32'd268435456  : 32'd1073741823],
60             [32'd1073741824 : 32'hFFFFFFFF]
61         };
62     }
63
64     constraint data_corner_c {
65         mem_wdata inside {32'd0, 32'd1, 32'd7, 32'd15, 32'd31, 32'd63, 32'd127, 32'd255, 32'd511,
66             32'd1023, 32'hFFFF_FFFF, 32'hAAAA_AAAA, 32'h5555_5555, 32'h1111_0000,
67             32'h0000_1111}; // Fixed: removed invalid constant

```

```

70     // Coverage group
71     coveragegroup Memory_cov;
72     option.per_instance = 1;
73
74     mem_en_cp : coverpoint mem_en {
75         bins disabled = {0};
76         bins enabled  = {1};
77     }
78
79     mem_we_cp : coverpoint mem_we iff (mem_en) {
80         bins read_op  = {0};
81         bins write_op = {1};
82     }
83
84     mem_addr_cp : coverpoint mem_addr iff (mem_en) {
85         bins corners[] = {10'd0, 10'd1, 10'd7, 10'd15, 10'd31, 10'd63, 10'd127, 10'd255,
86             10'd511, 10'd1023, 10'b1010101010, 10'b0101010101, 10'b1111111111,
87             10'b1111100000, 10'b0000011111};
88
89         bins all_values[] = {[10'd0 : 10'd1023]};
90     }

```

```

92     mem_wdata_cp : coverpoint mem_wdata iff (mem_en && mem_we) {
93         bins corners[] = {32'd0, 32'd1, 32'd7, 32'd15, 32'd31, 32'd63, 32'd127, 32'd255,
94             32'd511, 32'd1023, 32'hFFFF_FFFF, 32'hAAAA_AAAA, 32'h5555_5555,
95             32'h1111_0000, 32'h0000_1111};
96
97         bins grp1_1 = {[32'd0      : 32'd255]};
98         bins grp1_2 = {[32'd256    : 32'd1023]};
99         bins grp1_3 = {[32'd1024   : 32'd4095]};
100        bins grp1_4 = {[32'd4096   : 32'd16383]};
101        bins grp1_5 = {[32'd16384   : 32'd32767]};
102
103        bins grp2_1 = {[32'd32768   : 32'd49151]};
104        bins grp2_2 = {[32'd49152   : 32'd65535]};
105        bins grp2_3 = {[32'd65536   : 32'd262143]};
106        bins grp2_4 = {[32'd262144   : 32'd1048575]};
107
108        bins grp3_1 = {[32'd1048576   : 32'd16777215]};
109        bins grp3_2 = {[32'd16777216 : 32'd268435455]};
110        bins grp3_3 = {[32'd268435456 : 32'd1073741823]};
111        bins grp3_4 = {[32'd1073741824 : 32'hFFFFFFFF]};
112    }
113
114    endgroup
115    // Constructor
116    function new();
117        Memory_cov = new();
118    endfunction
119
120    // Post-randomize function with better formatting
121    function void display();
122        $display(">> post_randomize: en=%b, we=%b, addr=0x%03x(%0d), wdata=0x%08x",
123            mem_en, mem_we, mem_addr, mem_addr, mem_wdata);
124    endfunction
125 endclass

```

## 4. Test

```
1  `timescale 1ns/1ps
2  `include "memory_class.sv"
3
4  module mem_test (memory_interface.tb inter);
5      memory_class stim;
6      logic [31:0] golden_mem [0:1023];
7      logic [31:0] actual_data;
8      logic [31:0] expected_data;
9      int cases = 0;
10     int pass  = 0;
11     int fail  = 0;
12     int cov   = 0;
13
14     initial begin
15         // Initialize interface signals
16         inter.mem_en    = 0;
17         inter.mem_we    = 0;
18         inter.mem_addr  = 0;
19         inter.mem_wdata = 0;
20         inter.ARESETn   = 0;
21         cases           = 0;
22
23         // Create stimulus object
24         stim = new();
25
26         // Initialize golden memory
27         for (int i = 0; i < 1024; i++)
28             golden_mem[i] = 0;
29
30         // Reset sequence
31         repeat (2) @(posedge inter.ACLK);
32         inter.ARESETn = 1;
33         repeat (2) @(posedge inter.ACLK); // Allow reset to complete
```

```
36     stim.addr_c.constraint_mode(1);
37     stim.addr_corner_c.constraint_mode(0);
38     stim.data_c1.constraint_mode(1);
39     stim.data_c2.constraint_mode(0);
40     stim.data_c3.constraint_mode(0);
41     stim.data_corner_c.constraint_mode(0);
42     repeat (10000) begin
43         cases++;
44         generate_stimulus();
45         golden_model();
46         drive();
47         collect();
48         check();
49     end
50
51     stim.addr_c.constraint_mode(1);
52     stim.addr_corner_c.constraint_mode(0);
53     stim.data_c1.constraint_mode(0);
54     stim.data_c2.constraint_mode(1);
55     stim.data_c3.constraint_mode(0);
56     stim.data_corner_c.constraint_mode(0);
57     repeat (10000) begin
58         cases++;
59         generate_stimulus();
60         golden_model();
61         drive();
62         collect();
63         check();
64     end
```

```

66     stim.addr_c.constraint_mode(1);
67     stim.addr_corner_c.constraint_mode(0);
68     stim.data_c1.constraint_mode(0);
69     stim.data_c2.constraint_mode(0);
70     stim.data_c3.constraint_mode(1);
71     stim.data_corner_c.constraint_mode(0);
72     repeat (10000) begin
73         cases++;
74         generate_stimulus();
75         golden_model();
76         drive();
77         collect();
78         check();
79     end
80
81     stim.addr_c.constraint_mode(1);
82     stim.addr_corner_c.constraint_mode(0);
83     stim.data_c1.constraint_mode(0);
84     stim.data_c2.constraint_mode(0);
85     stim.data_c3.constraint_mode(0);
86     stim.data_corner_c.constraint_mode(1);
87     repeat (10000) begin
88         cases++;
89         generate_stimulus();
90         golden_model();
91         drive();
92         collect();
93         check();
94     end

```

```

96     stim.addr_c.constraint_mode(0);
97     stim.addr_corner_c.constraint_mode(1);
98     stim.data_c1.constraint_mode(1);
99     stim.data_c2.constraint_mode(0);
100    stim.data_c3.constraint_mode(0);
101    stim.data_corner_c.constraint_mode(0);
102    repeat (10000) begin
103        cases++;
104        generate_stimulus();
105        golden_model();
106        drive();
107        collect();
108        check();
109    end
110
111    stim.addr_c.constraint_mode(0);
112    stim.addr_corner_c.constraint_mode(1);
113    stim.data_c1.constraint_mode(0);
114    stim.data_c2.constraint_mode(1);
115    stim.data_c3.constraint_mode(0);
116    stim.data_corner_c.constraint_mode(0);
117    repeat (10000) begin
118        cases++;
119        generate_stimulus();
120        golden_model();
121        drive();
122        collect();
123        check();
124    end

```

```

126     stim.addr_c.constraint_mode(0);
127     stim.addr_corner_c.constraint_mode(1);
128     stim.data_c1.constraint_mode(0);
129     stim.data_c2.constraint_mode(0);
130     stim.data_c3.constraint_mode(1);
131     stim.data_corner_c.constraint_mode(0);
132     repeat (10000) begin
133         cases++;
134         generate_stimulus();
135         golden_model();
136         drive();
137         collect();
138         check();
139     end
140
141     stim.addr_c.constraint_mode(0);
142     stim.addr_corner_c.constraint_mode(1);
143     stim.data_c1.constraint_mode(0);
144     stim.data_c2.constraint_mode(0);
145     stim.data_c3.constraint_mode(0);
146     stim.data_corner_c.constraint_mode(1);
147     repeat (10000) begin
148         cases++;
149         generate_stimulus();
150         golden_model();
151         drive();
152         collect();
153         check();
154     end
155
156     cov = stim.Memory_cov.get_coverage();
157     $display("=== Final Results ===");
158     $display("Number of tests: %0d, Passed: %0d, Failed: %0d", cases, pass, fail);
159     $display("Coverage achieved: %0d%%", cov);

```

```

161     #200;
162     $finish;
163 end
164
165 task generate_stimulus();
166     if (!stim.randomize()) begin
167         $display("ERROR: Randomization failed at test case %0d", cases);
168         $finish;
169     end
170 endtask
171
172 task golden_model();
173     if (stim.mem_en) begin
174         if (stim.mem_we) begin
175             // Write operation
176             golden_mem[stim.mem_addr] = stim.mem_wdata;
177             expected_data = 'x; // Don't expect read data on write
178         end else begin
179             // Read operation
180             expected_data = golden_mem[stim.mem_addr];
181         end
182     end else begin
183         // Memory disabled
184         expected_data = 'x;
185     end
186 endtask

```



```

189 task drive();
190     // Drive stimulus to interface
191     inter.mem_en  = stim.mem_en;
192     inter.mem_we  = stim.mem_we;
193     inter.mem_addr = stim.mem_addr;
194     inter.mem_wdata = stim.mem_wdata;
195
196     // Sample coverage for stimulus
197     stim.Memory_cov.sample();
198
199     @(posedge inter.ACLK);
200 endtask
201
202 task collect();
203
204     if (stim.mem_en && !stim.mem_we)
205     begin
206         @(posedge inter.ACLK);
207         actual_data = inter.mem_rdata;
208     end else
209     begin
210         actual_data = 'x;
211     end
212 endtask

```

```

214 task check();
215     if (stim.mem_en && !stim.mem_we)
216     begin
217         if (actual_data != expected_data)
218         begin
219             $display("FAIL [%0d]: Addr=0x%03x, Expected=0x%08x, Got=0x%08x",
220                 cases, stim.mem_addr, expected_data, actual_data);
221             fail++;
222         end else
223         begin
224             pass++;
225             $display("PASS [%0d]: Addr=0x%03x, Data=0x%08x",
226                 cases, stim.mem_addr, actual_data);
227         end
228     end else if (stim.mem_en && stim.mem_we)
229     begin
230         $display("WRITE [%0d]: Addr=0x%03x, Data=0x%08x",
231             cases, stim.mem_addr, stim.mem_wdata);
232     end
233     $display("");
234 endtask
235 endmodule

```

## 5. Assertions

```
1  module mem_assert (memory_interface.dut inter);
2
3      // Property 1: Write operations
4      property Write_prop;
5          @(posedge inter.ACLK)
6              disable iff (!inter.ARESETn)
7              inter.mem_we |-> inter.mem_en; // If mem_we is 1 , mem_en is must be 1
8      endproperty
9
10     A1: assert property (Write_prop)
11         | else $error("Write operation failed");
12     C1: cover property (Write_prop);
13
14     // Property 2: Read operation
15     property Read_prop;
16         @(posedge inter.ACLK)
17             disable iff (!inter.ARESETn)
18             (inter.mem_en && !inter.mem_we) |>= !$isunknown(inter.mem_rdata);
19     endproperty
20
21     A2: assert property (Read_prop)
22         | else $error("Read operation failed");
23     C2: cover property (Read_prop);
24
25     // Property 3: Write enable
26     property enable_check;
27         @(posedge inter.ACLK)
28             disable iff (!inter.ARESETn)
29             !inter.mem_en |-> !inter.mem_we;
30     endproperty
31
32     A3: assert property (enable_check)
33         | else $error("mem_we asserted when mem_en is low");
34     C3: cover property (enable_check);
35
```

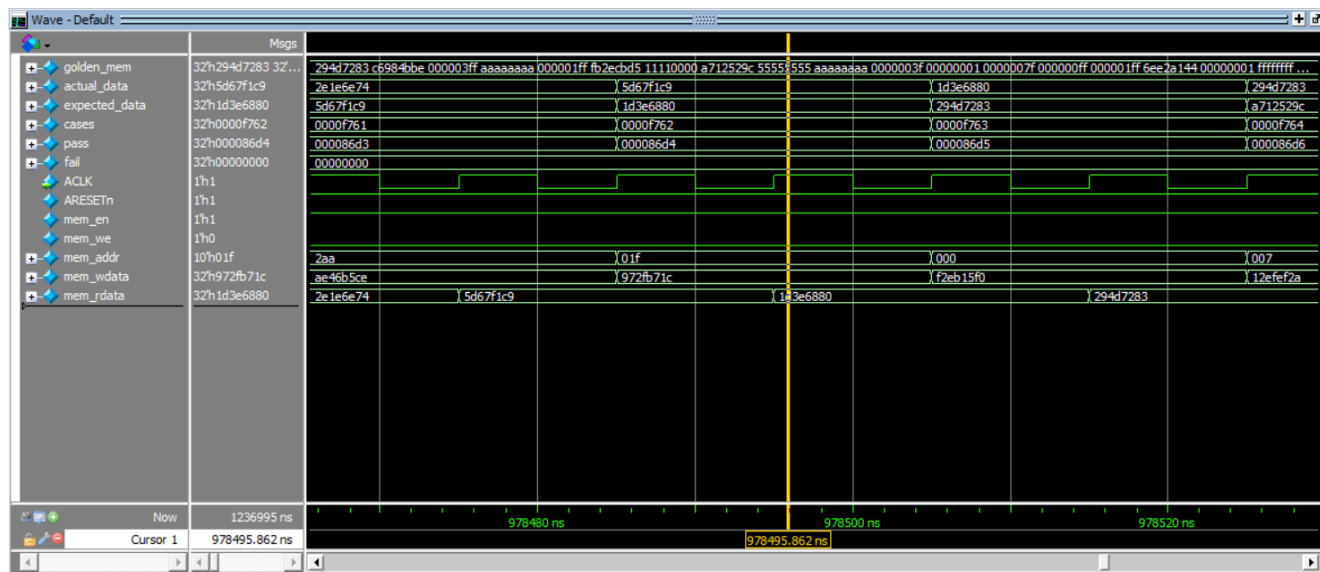
```
36     // Property 4: Reset
37     property reset_prop;
38         @(posedge inter.ACLK)
39             !inter.ARESETn |>= (inter.mem_rdata == 0);
40     endproperty
41
42     A4: assert property (reset_prop)
43         | else $error("Reset protocol failed");
44     C4: cover property (reset_prop);
45
46     // Property 5: Memory operations
47     property stable_signals;
48         @(posedge inter.ACLK)
49             disable iff (!inter.ARESETn)
50             inter.mem_en |-> (inter.mem_we == 1'b0 || inter.mem_we == 1'b1);
51     endproperty
52
53     A5: assert property (stable_signals)
54         | else $error("Invalid control signal values");
55     C5: cover property (stable_signals);
56
57     // Property 6: Address should be within valid range
58     property valid_address;
59         @(posedge inter.ACLK)
60             disable iff (!inter.ARESETn)
61             inter.mem_en |-> (inter.mem_addr < 1024);
62     endproperty
63
64     A6: assert property (valid_address)
65         | else $error("Invalid address");
66     C6: cover property (valid_address);
67
68     endmodule
```

## II. Memory Results

```
#
# === Final Results ===
# Number of tests: 80000, Passed: 43676, Failed: 0
# Coverage achieved: 100%
# ** Note: $finish      : mem_test.sv(164)
#   Time: 1236995 ns   Iteration: 0   Instance: /top/tb
# 1
```

- **Comment:** Passed cases represent the cases that, when mem\_en is asserted and the operation is read.

Cover Directives															
Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads	
/top/check/C1	SVA	✓	Off	28333	1	Unli...	1	100%	████████	✓	0	0	0 ps	0	
/top/check/C2	SVA	✓	Off	87370	1	Unli...	1	100%	████████	✓	0	0	0 ps	0	
/top/check/C3	SVA	✓	Off	7994	1	Unli...	1	100%	████████	✓	0	0	0 ps	0	
/top/check/C4	SVA	✓	Off	2	1	Unli...	1	100%	████████	✓	0	0	0 ps	0	
/top/check/C5	SVA	✓	Off	15704	1	Unli...	1	100%	████████	✓	0	0	0 ps	0	
/top/check/C6	SVA	✓	Off	15704	1	Unli...	1	100%	████████	✓	0	0	0 ps	0	



## Function coverage

Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances
/mem_test_sv_unit/memory_class		100.00%					
TYPE Memory_cov		100.00%	100	100.00...		✓	auto(1)
CVP Memory_cov::mem_en_cp		100.00%	100	100.00...		✓	
CVP Memory_cov::mem_we_cp		100.00%	100	100.00...		✓	
CVP Memory_cov::mem_addr_cp		100.00%	100	100.00...		✓	
CVP Memory_cov::mem_wdata_cp		100.00%	100	100.00...		✓	
INST Vmem_test_sv_unit::memory_class::Memory_cov		100.00%	100	100.00...		✓	

## Code coverage

```
=====  
=== Instance: /top/inter  
=== Design Unit: work.memory_interface  
=====
```

Toggle Coverage:				
Enabled Coverage	Bins	Hits	Misses	Coverage
-----				
Toggles	156	155	1	99.35%

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

Toggle Coverage for instance /top/inter --

Node	1H->0L	0L->1H	"Coverage"
-----			
ARESETn	0	1	50.00

```
Total Node Count      =      78  
Toggled Node Count    =      77  
Untoggled Node Count  =       1  
  
Toggle Coverage       =    99.35% (155 of 156 bins)  
=====
```

- **Comment:** We do reset only one at the beginning.

```
=====  
=== Instance: /top/dut  
=== Design Unit: work.axi4_memory  
=====
```

Branch Coverage:				
Enabled Coverage	Bins	Hits	Misses	Coverage
-----				
Branches	5	5	0	100.00%

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

Branch Coverage for instance /top/dut

Line	Item	Count	Source
-----			
File axi_memory.sv			
-----IF Branch-----			
16		123699	Count coming in to IF
16	1	2	
18	1	115704	
		7993	All False Count
Branch totals: 3 hits of 3 branches = 100.00%			
-----IF Branch-----			
19		115704	Count coming in to IF
19	1	28333	
21	1	87371	
Branch totals: 2 hits of 2 branches = 100.00%			

```
Statement Coverage:
  Enabled Coverage
  -----
  Statements      7      7      0  100.00%

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

Statement Coverage for instance /top/dut --

  Line      Item      Count      Source
  ----      -
File axi_memory.sv
15          1      123699
17          1          2
20          1      28333
22          1      87371
28          1          1
28          2      1024
29          1      1024
Toggle Coverage:
  Enabled Coverage
  -----
  Toggles        64      0      64  0.00%

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

# III. System

## 1. Interface

```
1  interface axi4_interface (input bit ACLK);
2      // axi4 signals
3      bit          ARESETn;
4      logic        AWVALID;
5      logic        AWREADY;
6      logic        WLAST;
7      logic        RLAST;
8      logic        WVALID;
9      logic        WREADY;
10     logic        RVALID;
11     logic        RREADY;
12     logic        ARREADY;
13     logic        ARVALID;
14     logic        BVALID;
15     logic        BREADY;
16     logic [15:0]  AWADDR;
17     logic [7:0]   AWLEN;
18     logic [2:0]   AWSIZE;
19     logic [31:0]  WDATA;
20     logic [15:0]  ARADDR;
21     logic [31:0]  RDATA;
22     logic [7:0]   ARLEN;
23     logic [2:0]   ARSIZE;
24     logic [1:0]   RRESP;
25     logic [1:0]   BRESP;
26
```

```
28  modport dut (
29      input  ACLK,
30      input  ARESETn,
31
32      // Write address channel
33      input  AWADDR,
34      input  AWLEN,
35      input  AWSIZE,
36      input  AWVALID,
37      output AWREADY,
38
39      // Write data channel
40      input  WDATA,
41      input  WVALID,
42      input  WLAST,
43      output WREADY,
44
45      // Write response channel
46      output BRESP,
47      output BVALID,
48      input  BREADY,
49
50      // Read address channel
51      input  ARADDR,
52      input  ARLEN,
53      input  ARSIZE,
54      input  ARVALID,
55      output ARREADY,
56
57      // Read data channel
58      output RDATA,
59      output RRESP,
60      output RVALID,
61      output RLAST,
62      input  RREADY );
```

```

64  modport tb (
65      input  ACLK,
66      output ARESETn,
67
68      // Write address channel
69      output AWADDR,
70      output AWLEN,
71      output AWSIZE,
72      output AWVALID,
73      input  AWREADY,
74
75      // Write data channel
76      output WDATA,
77      output WVALID,
78      output WLAST,
79      input  WREADY,
80
81      // Write response channel
82      input  BRESP,
83      input  BVALID,
84      output BREADY,
85
86      // Read address channel
87      output ARADDR,
88      output ARLEN,
89      output ARSIZE,
90      output ARVALID,
91      input  ARREADY,
92
93      // Read data channel
94      input  RDATA,
95      input  RRESP,
96      input  RVALID,
97      input  RLAST,
98      output RREADY );
99  endinterface

```

```

1  `include "axi4_interface.sv"
2  `include "Axi4_test.sv"
3  `include "axi4.sv"
4  module top();
5
6      // axi4 signals
7      bit      ARESETn;
8      logic     AWVALID;
9      logic     AWREADY;
10     logic     WLAST;
11     logic     RLAST;
12     logic     WVALID;
13     logic     WREADY;
14     logic     RVALID;
15     logic     RREADY;
16     logic     ARREADY;
17     logic     ARVALID;
18     logic     BVALID;
19     logic     BREADY;
20     logic [15:0] AWADDR;
21     logic [7:0]  AWLEN;
22     logic [2:0]  AWSIZE;
23     logic [31:0] WDATA;
24     logic [31:0] ARADDR;
25     logic [31:0] RDATA;
26     logic [7:0]  ARLEN;
27     logic [2:0]  ARSIZE;
28     logic [1:0]  RRESP;
29     logic [1:0]  BRESP;
30
31     bit ACLK ;
32     initial begin
33         ACLK = 0;
34         forever begin
35             #5ns ACLK = ~ACLK;
36         end
37     end

```

```

39     axi4_interface inter (ACLK);
40     axi4             dut  (inter.dut);
41     Axi4_test        tb   (inter.tb);
42     axi4_assert      check (inter.dut);
43
44     endmodule

```

2. **Design bug:** When we randomize, this case happens and forces us to check the design again, as it is unexpected, so we discover this bug

```

# AWADDR = 3063, AWVALID = 1, AWLEN = 244, AWSIZE = 2, WVALID = 1
# Test 12 FAILED  BRESP = SLVERR

```

```

// Address boundary check (4KB boundary = 12 bits)
assign write_boundary_cross = ((write_addr & 12'hFFF) + (write_burst_len << write_size)) > 12'hFFF;
assign read_boundary_cross  = ((read_addr & 12'hFFF) + (read_burst_len << read_size)) > 12'hFFF;

```

```

// Address boundary check (4KB boundary = 12 bits)
wire write_boundary_cross = ((inter.AWADDR & 12'hFFF) + ((inter.AWLEN + 1) << inter.AWSIZE)) > 12'hFFF;
wire read_boundary_cross  = ((inter.ARADDR & 12'hFFF) + ((inter.ARLen + 1) << inter.ARSIZE)) > 12'hFFF;

```

- **Comment:** We change write\_addr by AWADDR because when we check, we must check by the first address, not the current address or the final address, and we change AWLEN by AWLEN + 1 ... and the same thing for read operation.



### 3. Class

```
1  class axi4_class;
2
3  // Write signals
4  rand logic [15:0] ADDR;
5  rand logic [7:0] LEN;
6  rand logic [2:0] SIZE;
7  rand logic [31:0] DATA;
8  rand logic [1:0] OPERATION;
9
10 // Random delays for handshaking
11 rand logic [2:0] aw_valid_delay;
12 rand logic [2:0] w_valid_delay;
13 rand logic [2:0] b_ready_delay;
14 rand logic [2:0] ar_valid_delay;
15 rand logic [2:0] r_ready_delay;
16
17 // Constraints
18 constraint OPERATION_C {
19     OPERATION inside {[2'd1:2'd2]};
20 }
21
22 // Delay constraints
23 constraint delay_ranges {
24     aw_valid_delay inside {[0:7]};
25     w_valid_delay inside {[0:7]};
26     b_ready_delay inside {[0:7]};
27     ar_valid_delay inside {[0:7]};
28     r_ready_delay inside {[0:7]};
29 }
30
31 constraint fix_size {
32     SIZE == 2; // Fixed to 2 (4 bytes)
33 }
34
35 constraint aligned_address {
36     ADDR[1:0] == 2'b00; // Aligned to 4-byte boundary
37 }
38
39 // Valid access constraint
40 constraint valid_range {
41     ((ADDR >> 2) + (LEN + 1)) < 1024;
42 }
43
44 // Invalid access constraints
45 constraint invalid_access_c {
46     ((ADDR >> 2) + (LEN + 1)) >= 1024;
47 }
48
49 constraint LEN_range_c {
50     LEN inside {[8'd0 : 8'd255]};
51 }
52
53 constraint LEN_corners_c {
54     LEN inside {8'd0, 8'd1, 8'd127, 8'd128, 8'd254, 8'd255};
55 }
56
57 constraint addr_c {
58     ADDR inside {[16'd0 : 16'd65535]};
59 }
60
61 constraint addr_corner_c {
62     ADDR inside {16'd0, 16'd1024, 16'd2048, 16'd4092, 16'b1111_1111_1100}; // All 4-byte aligned
63 }
64
65
```

```

66 constraint data_c1 {
67     DATA inside {
68         [32'd0           : 32'd255],
69         [32'd256        : 32'd1023],
70         [32'd1024       : 32'd4095],
71         [32'd4096       : 32'd16383],
72         [32'd16384      : 32'd32767]
73     };
74 }
75
76 constraint data_c2 {
77     DATA inside {
78         [32'b0          : 32'b1],
79         [32'd32768      : 32'd49151],
80         [32'd49152      : 32'd65535],
81         [32'd65536      : 32'd262143],
82         [32'd262144     : 32'd1048575]
83     };
84 }
85
86 constraint data_c3 {
87     DATA inside {
88         [32'b0          : 32'b1],
89         [32'd1048576     : 32'd16777215],
90         [32'd16777216    : 32'd268435455],
91         [32'd268435456   : 32'd1073741823],
92         [32'd1073741824  : 32'hFFFFFFF]
93     };
94 }
95
96 constraint data_corner_c {
97     DATA inside {32'd0, 32'd1, 32'hFFFF_FFFF, 32'hAAAA_AAAA, 32'h1111_0000, 32'h0000_1111};
98 }
99

```

```

100 // Coverage
101 covergroup axi4_cov;
102
103     // LEN coverage with corner bins
104     coverpoint LEN {
105         bins corner0 = {8'd0};
106         bins corner1 = {8'd1};
107         bins corner2 = {8'd127};
108         bins corner3 = {8'd128};
109         bins corner4 = {8'd254};
110         bins corner5 = {8'd255};
111         bins auto_bins[] = {[8'd0:8'd255]};
112     }
113
114     // ADDR coverage with corner bins
115     coverpoint ADDR {
116         bins corner0 = {16'd0};
117         bins corner1 = {16'd1024};
118         bins corner2 = {16'd2048};
119         bins corner3 = {16'd4092};
120         bins corner4 = {16'b1111_1111_1100};
121         bins range_0 = { [16'd0       : 16'd255] };
122         bins range_1 = { [16'd256     : 16'd1023] };
123         bins range_2 = { [16'd1024    : 16'd4095] };
124         bins range_3 = { [16'd4096    : 16'd16383] };
125         bins range_4 = { [16'd16384   : 16'd32767] };
126         bins range_5 = { [16'd32768   : 16'd49151] };
127         bins range_6 = { [16'd49152   : 16'd65535] };
128     }
129

```

```

130 // Fixed size coverage
131 coverpoint SIZE {
132     bins fixed_size = {2};
133     illegal_bins others = default;
134 }
135
136 // DATA coverage with corner bins
137 coverpoint DATA {
138     bins corner0      = {32'd0};
139     bins corner1      = {32'd1};
140     bins corner2      = {32'hFFFF_FFFF};
141     bins corner3      = {32'hAAAA_AAAA};
142     bins corner4      = {32'h1111_0000};
143     bins corner5      = {32'h0000_1111};
144     bins range_0       = { [32'd0      : 32'd255] };
145     bins range_1       = { [32'd256    : 32'd1023] };
146     bins range_2       = { [32'd1024   : 32'd4095] };
147     bins range_3       = { [32'd4096   : 32'd16383] };
148     bins range_4       = { [32'd16384   : 32'd32767] };
149     bins range_5       = { [32'd32768   : 32'd49151] };
150     bins range_6       = { [32'd49152   : 32'd65535] };
151     bins range_7       = { [32'd65536   : 32'd262143] };
152     bins range_8       = { [32'd262144   : 32'd1048575] };
153     bins range_9       = { [32'd1048576 : 32'd16777215] };
154     bins range_10      = { [32'd16777216 : 32'd268435455] };
155     bins range_11      = { [32'd268435456 : 32'd1073741823] };
156     bins range_12      = { [32'd1073741824 : 32'hFFFFFFFF] };
157 }
158
159 // Memory access bounds coverage
160 coverpoint ((ADDR >> 2) + (LEN + 1)) {
161     bins valid_access  = {[0:1024]}; // Fits within memory
162     bins invalid_access = {[1025:$]}; // Exceeds memory
163 }
164

```

```

165 // Delay coverage
166 coverpoint aw_valid_delay { bins all_values[] = {[0:7]}; }
167 coverpoint w_valid_delay { bins all_values[] = {[0:7]}; }
168 coverpoint b_ready_delay { bins all_values[] = {[0:7]}; }
169 coverpoint ar_valid_delay { bins all_values[] = {[0:7]}; }
170 coverpoint r_ready_delay { bins all_values[] = {[0:7]}; }
171
172 // Cross coverage
173 cross LEN, ADDR;
174 cross DATA, LEN;
175 cross DATA, ADDR;
176 cross aw_valid_delay, w_valid_delay;
177 cross ar_valid_delay, r_ready_delay;
178
179 endgroup
180
181 function void display();
182     $display("ADDR = %0d, LEN = %0d, SIZE = %0d, Memory Access = %0d",
183         ADDR, LEN, SIZE, ((ADDR >> 2) + (LEN + 1)));
184     $display("Delays - AW:%0d, W:%0d, B:%0d, AR:%0d, R:%0d",
185         aw_valid_delay, w_valid_delay, b_ready_delay, ar_valid_delay, r_ready_delay);
186 endfunction
187
188 function new();
189     axi4_cov = new();
190 endfunction
191
192 endclass

```

## 4. Test

```
1  `timescale 1ns/1ps
2  `include "axi4_class.sv"
3  `include "axi4_interface.sv"
4
5  module Axi4_test (axi4_interface.tb inter);
6
7      axi4_class stim;
8      logic [31:0] golden_mem [0:1023];
9      logic [31:0] expected_queue[$];
10     logic [31:0] actual_queue[$];
11     logic [31:0] data[$];
12     logic [1:0] expected_response;
13     logic [1:0] actual_response;
14
15     int cases = 0;
16     int pass = 0;
17     int fail = 0;
18     real cov = 0;
19
20
21     bit range_mode [[2] = '{ '1,0}, '{0,1} }; // valid, invalid range
22     bit awlen_modes [[2] = '{ '1,0}, '{0,1} }; // range, corners
23     bit addr_modes [[2] = '{ '1,0}, '{0,1} }; // range, corners
24     bit data_modes [[4] = '{ '1,0,0,0}, '{0,1,0,0}, '{0,0,1,0}, '{0,0,0,1} };
25
26     initial begin
27         stim = new();
28
29         // Initialize golden memory
30         for (int i = 0; i < 1024; i++)
31             golden_mem[i] = 0;
32
33         actual_queue.delete();
34         expected_queue.delete();
35         data.delete();
36         reset();
37     end
```

```

38     stim.delay_ranges.constraint_mode(1);
39     stim.fix_size.constraint_mode(1);
40     stim.aligned_address.constraint_mode(1);
41
42     foreach (range_mode[m]) begin
43         foreach (awlen_modes[i]) begin
44             foreach (addr_modes[j]) begin
45                 foreach (data_modes[k]) begin
46
47                     stim.valid_range.constraint_mode(range_mode[m][0]);
48                     stim.invalid_access_c.constraint_mode(range_mode[m][1]);
49
50                     stim.LEN_range_c.constraint_mode(awlen_modes[i][0]);
51                     stim.LEN_corners_c.constraint_mode(awlen_modes[i][1]);
52
53                     stim.addr_c.constraint_mode(addr_modes[j][0]);
54                     stim.addr_corner_c.constraint_mode(addr_modes[j][1]);
55
56                     stim.data_c1.constraint_mode(data_modes[k][0]);
57                     stim.data_c2.constraint_mode(data_modes[k][1]);
58                     stim.data_c3.constraint_mode(data_modes[k][2]);
59                     stim.data_corner_c.constraint_mode(data_modes[k][3]);
60
61                     repeat (5100) begin
62                         cases++;
63                         clear_signals();
64                         generate_stimulus();
65                         golden_model();
66                         drive();
67                     end
68                 end
69             end
70         end
71     end

```

```

72
73     cov = stim.axi4_cov.get_coverage();
74     $display("=== Final Results ===");
75     $display("Number of tests: %0d, Passed: %0d, Failed: %0d", cases, pass, fail);
76     $display("Coverage achieved: %0.1f%%", cov);
77
78     #200;
79     $stop;
80 end
81
82 task reset();
83     begin
84         inter.ARESETn = 1'b1;
85         @(negedge inter.ACLK);
86         inter.ARESETn = 1'b0;
87         @(negedge inter.ACLK);
88         inter.ARESETn = 1'b1;
89         @(negedge inter.ACLK);
90     end
91 endtask
92

```

```

93     task clear_signals();
94         inter.AWVALID = 0;
95         inter.WVALID  = 0;
96         inter.WLAST   = 0;
97         inter.BREADY  = 0;
98         inter.ARVALID = 0;
99         inter.RREADY  = 0;
100        inter.AWADDR = 0;
101        inter.AWLEN  = 0;
102        inter.AWSIZE = 0;
103        inter.WDATA  = 0;
104        inter.ARADDR = 0;
105        inter.ARLLEN = 0;
106        inter.ARSIZE = 0;
107        expected_response = 2'b0;
108        actual_response   = 2'b0;
109        @(negedge inter.ACLK);
110    endtask
111
112    task generate_stimulus();
113        data.delete();
114        stim.LEN      = 0;
115        stim.ADDR     = 0;
116        stim.SIZE     = 0;
117        stim.DATA     = 0;
118        stim.OPERATION = 0;
119        stim.aw_valid_delay = 0;
120        stim.w_valid_delay = 0;
121        stim.b_ready_delay = 0;
122        stim.ar_valid_delay = 0;
123        stim.r_ready_delay = 0;
124
125        assert(stim.randomize(ADDR, LEN, SIZE, OPERATION, aw_valid_delay, w_valid_delay,
126            | b_ready_delay, ar_valid_delay, r_ready_delay ))
127            else $fatal("Address/Length/Operation randomization failed");
128    endtask

```

```

129        if (stim.OPERATION == 2'd1)
130            begin
131                for (int i = 0; i <= stim.LEN; i++)
132                    begin
133                        assert(stim.randomize(DATA))
134                            else $fatal("Data randomization failed for beat %0d", i);
135                        data[i] = stim.DATA;
136                        stim.axi4_cov.sample();
137                    end
138            end
139    endtask
140

```

```

141 task golden_model();
142     expected_queue.delete();
143     if (((stim.ADDR >> stim.SIZE) + (stim.LEN + 1)) >= 1024)
144     begin
145         expected_response = 2'b10;
146     end else
147     begin
148         expected_response = 2'b00;
149         if (stim.OPERATION == 2'd1)
150         begin
151             for (int i = 0; i <= stim.LEN; i++)
152             begin
153                 golden_mem[(stim.ADDR >> 2) + i] = data[i];
154             end
155         end else if (stim.OPERATION == 2'd2)
156         begin
157             for (int i = 0; i <= stim.LEN; i++)
158             begin
159                 expected_queue[i] = golden_mem[(stim.ADDR >> 2) + i];
160             end
161         end
162     end
163 endtask
164

```

```

165 task drive();
166     if (stim.OPERATION == 2'd2)
167     begin
168         // Apply delay before starting read address phase
169         repeat (stim.ar_valid_delay) @(negedge inter.ACLK);
170
171         // Set up read address channel
172         inter.ARADDR = stim.ADDR;
173         inter.ARLLEN = stim.LEN;
174         inter.ARSIZE = stim.SIZE;
175         inter.ARVALID = 1'b1;
176
177         // Wait for address acceptance
178         repeat (20) @(negedge inter.ACLK)
179         | if (inter.ARREADY) break;
180         inter.ARVALID = 0;
181
182         collect();
183     end else if (stim.OPERATION == 2'd1)
184     begin
185         // Apply delay before starting write address phase
186         repeat (stim.aw_valid_delay) @(negedge inter.ACLK);
187         // Start write address transaction
188
189         inter.AWADDR = stim.ADDR;
190         inter.AWLEN = stim.LEN;
191         inter.AWSIZE = stim.SIZE;
192         inter.AWVALID = 1'b1;
193
194         // Wait for address acceptance
195         repeat (20) @(negedge inter.ACLK)
196         | if (inter.AWREADY) break;
197         inter.AWVALID = 1'b0;
198
199

```

```

200 if (((stim.ADDR >> stim.SIZE) + (stim.LEN + 1)) >= 1024)
201 begin
202     $display("TIME: $0t", $time);
203     assert(stim.randomize(w_valid_delay))
204     | else $fatal("w_valid_delay randomization failed");
205     repeat (stim.w_valid_delay) @(negedge inter.ACLK);
206     stim.axi4_cov.sample();
207     inter.WVALID = 1'b1;
208     inter.WLAST = 1;
209     //@(negedge inter.ACLK);
210
211     // Wait for write data acceptance
212     repeat (20) @(negedge inter.ACLK)
213     if (inter.WREADY) break;
214
215     actual_response = inter.BRESP;
216     inter.WVALID = 1'b0;
217     inter.WLAST = 0;
218
219     // Apply delay before accepting write response
220     repeat (stim.b_ready_delay) @(negedge inter.ACLK);
221     inter.BREADY = 1'b1;
222     repeat (20) @(negedge inter.ACLK)
223     if (inter.BVALID) break;
224     inter.BREADY = 0;
225 end else
226 begin
227
228     // Apply delay before starting write data phase
229     repeat (stim.w_valid_delay) @(negedge inter.ACLK);
230

```

```

231 // Send burst data with proper handshaking
232 for (int i = 0; i <= stim.LEN; i++)
233 begin
234     inter.WDATA = data[i];
235     inter.WVALID = 1'b1;
236     inter.WLAST = (i == stim.LEN);
237
238     // Wait for write data acceptance
239     repeat (20) @(negedge inter.ACLK)
240     | if (inter.WREADY) break;
241
242     inter.WVALID = 1'b0;
243     // Generate random delay for each write data beat
244     assert(stim.randomize(w_valid_delay))
245     | else $fatal("w_valid_delay randomization failed");
246     repeat (stim.w_valid_delay) @(negedge inter.ACLK);
247     stim.axi4_cov.sample();
248 end
249 actual_response = inter.BRESP;
250 inter.WLAST = 0;
251
252 // Apply delay before accepting write response
253 repeat (stim.b_ready_delay) @(negedge inter.ACLK);
254 inter.BREADY = 1'b1;
255
256 // Wait for write response
257 repeat (20) @(negedge inter.ACLK)
258 | if (inter.BVALID) break;
259
260 inter.BREADY = 0;
261 end
262 check();
263 end
264 endtask

```



```

266 task collect();
267     actual_queue.delete();
268     // Start with RREADY high, then apply delay by temporarily deasserting
269     inter.RREADY = 1'b1;
270
271     for (int i = 0; i <= stim.LEN; i++)
272     begin
273         // Apply r_ready_delay for each read beat
274         if (stim.r_ready_delay > 0)
275         begin
276             inter.RREADY = 1'b0;
277             repeat (stim.r_ready_delay) @(negedge inter.ACLK);
278             inter.RREADY = 1'b1;
279         end
280
281         // Generate random delay for each read beat
282         assert(stim.randomize(r_ready_delay))
283         | else $fatal("r_ready_delay randomization failed");
284         stim.axi4_cov.sample();
285
286         if (i == stim.LEN && !inter.RLAST) begin
287             $warning("RLAST not asserted on final beat");
288         end
289
290         // Wait for valid read data
291         repeat (20) @(negedge inter.ACLK)
292         | if (inter.RVALID) break;
293
294         actual_queue.push_back(inter.RDATA);
295     end
296     actual_response = inter.RRESP;
297     inter.RREADY = 1'b0;
298     check();
299 endtask
300

```

```

301 task check();
302     if (stim.OPERATION == 2'd1)
303     begin
304         $display("Test %0d", cases);
305         $display("Write Test");
306         stim.display();
307
308         if (actual_response == expected_response)
309         begin
310             pass++;
311             $display("Test %0d PASSED", cases);
312             $display("write Operation passed");
313         end else
314         begin
315             fail++;
316             $display("Test %0d FAILED Resopnse mismatch", cases);
317             $display("write Operation failed");
318         end
319
320     end else
321     if (stim.OPERATION == 2'd2)
322     begin
323         $display("Test %0d", cases);
324         $display("Read Test");
325         stim.display();
326

```

```

327 if (((stim.ADDR >> stim.SIZE) + (stim.LEN + 1)) >= 1024)
328 begin
329     if (actual_response == expected_response)
330     begin
331         pass++;
332         $display("Test %0d PASSED", cases);
333         $display("Read Operation passed");
334         $display("Expected response: %2b, Actual response: %2b",
335             expected_response, actual_response);
336     end else
337     begin
338         fail++;
339         $display("Test %0d FAILED - Data|Resopnse mismatch", cases);
340         $display("Read Operation failed");
341         $display("Expected response: %2b, Actual response: %2b",
342             expected_response, actual_response);
343     end
344     $display("");
345     return;
346 end else
347 begin
348
349     if (actual_queue.size() != expected_queue.size())
350     begin
351         fail++;
352         $display("Test %0d FAILED - Queue size mismatch", cases);
353         $display("Read Operation failed");
354         $display("Expected size: %0d, Actual size: %0d",
355             expected_queue.size(), actual_queue.size());
356         $display("");
357         return;
358     end

```

```

360 if ((actual_queue == expected_queue) && (actual_response == expected_response))
361 begin
362     pass++;
363     $display("Test %0d PASSED", cases);
364     $display("Read Operation passed");
365     $display("Expected response: %2b, Actual response: %2b",
366         expected_response, actual_response);
367
368 end else
369 begin
370     fail++;
371     $display("Test %0d FAILED - Data|Resopnse mismatch", cases);
372     $display("Read Operation failed");
373     $display("Expected response: %2b, Actual response: %2b",
374         expected_response, actual_response);
375
376     if (actual_response != expected_response)
377     begin
378         $display("Expected response: %2b, Actual response: %2b",
379             expected_response, actual_response);
380     end
381

```

```

382         for (int i = 0; i < expected_queue.size(); i++)
383         begin
384             if (expected_queue[i] != actual_queue[i])
385             begin
386                 $display(" Beat[%0d]: Expected = %h, Actual = %h",
387                     i, expected_queue[i], actual_queue[i]);
388             end
389         end
390     end
391 end
392 end
393 $display("");
394 endtask
395 endmodule

```

## 5. Assertions

```

1  module axi4_assert (axi4_interface.dut inter);
2
3  // All outputs should be properly initialized after reset
4  property reset_awready;
5      @(posedge inter.ACLK) !inter.ARESETn |> inter.AWREADY == 1'b1;
6  endproperty
7  A_RESET_AWREADY: assert property (reset_awready)
8      else $error("AWREADY not initialized to 1 after reset");
9  C_RESET_AWREADY: cover property (reset_awready);
10
11  property reset_wready;
12      @(posedge inter.ACLK) !inter.ARESETn |> inter.WREADY == 1'b0;
13  endproperty
14  A_RESET_WREADY: assert property (reset_wready)
15      else $error("WREADY not initialized to 0 after reset");
16  C_RESET_WREADY: cover property (reset_wready);
17
18  property reset_bvalid;
19      @(posedge inter.ACLK) !inter.ARESETn |> inter.BVALID == 1'b0;
20  endproperty
21  A_RESET_BVALID: assert property (reset_bvalid)
22      else $error("BVALID not initialized to 0 after reset");
23  C_RESET_BVALID: cover property (reset_bvalid);
24
25  property reset_arready;
26      @(posedge inter.ACLK) !inter.ARESETn |> inter.ARREADY == 1'b1;
27  endproperty
28  A_RESET_ARREADY: assert property (reset_arready)
29      else $error("ARREADY not initialized to 1 after reset");
30  C_RESET_ARREADY: cover property (reset_arready);
31
32  property reset_rvalid;
33      @(posedge inter.ACLK) !inter.ARESETn |> inter.RVALID == 1'b0;
34  endproperty
35  A_RESET_RVALID: assert property (reset_rvalid)
36      else $error("RVALID not initialized to 0 after reset");
37  C_RESET_RVALID: cover property (reset_rvalid);

```

```

39 property reset_rlast;
40 | @(posedge inter.ACLK) !inter.ARESETn |-> inter.RLAST == 1'b0;
41 endproperty
42 A_RESET_RLAST: assert property (reset_rlast)
43 | else $error("RLAST not initialized to 0 after reset");
44 C_RESET_RLAST: cover property (reset_rlast);
45
46 // AWREADY should go low after accepting address
47 property awready_deassert;
48 | @(posedge inter.ACLK) disable iff (!inter.ARESETn)
49 | (inter.AWVALID && inter.AWREADY) |=> !inter.AWREADY;
50 endproperty
51 A_AWREADY_DEASSERT: assert property (awready_deassert)
52 | else $error("AWREADY should deassert after address handshake");
53 C_AWREADY_DEASSERT: cover property (awready_deassert);
54
55 // Address signals should remain stable when AWVALID is high
56 property awaddr_stable;
57 | @(posedge inter.ACLK) disable iff (!inter.ARESETn)
58 | inter.AWVALID && !inter.AWREADY |=> $stable(inter.AWADDR);
59 endproperty
60 A_AWADDR_STABLE: assert property (awaddr_stable)
61 | else $error("AWADDR must remain stable when AWVALID is high");
62 C_AWADDR_STABLE: cover property (awaddr_stable);
63
64 property awlen_stable;
65 | @(posedge inter.ACLK) disable iff (!inter.ARESETn)
66 | inter.AWVALID && !inter.AWREADY |=> $stable(inter.AWLEN);
67 endproperty
68 A_AWLEN_STABLE: assert property (awlen_stable)
69 | else $error("AWLEN must remain stable when AWVALID is high");
70 C_AWLEN_STABLE: cover property (awlen_stable);

```

```

72 property awsize_stable;
73 | @(posedge inter.ACLK) disable iff (!inter.ARESETn)
74 | inter.AWVALID && !inter.AWREADY |=> $stable(inter.AWSIZE);
75 endproperty
76 A_AWSIZE_STABLE: assert property (awsize_stable)
77 | else $error("AWSIZE must remain stable when AWVALID is high");
78 C_AWSIZE_STABLE: cover property (awsize_stable);
79
80 // WDATA should remain stable when WVALID is high
81 property wdata_stable;
82 | @(posedge inter.ACLK) disable iff (!inter.ARESETn)
83 | inter.WVALID && !inter.WREADY |=> $stable(inter.WDATA);
84 endproperty
85 A_WDATA_STABLE: assert property (wdata_stable)
86 | else $error("WDATA must remain stable when WVALID is high");
87 C_WDATA_STABLE: cover property (wdata_stable);
88
89 // WLAST should be asserted on the last data beat
90 property wlast_on_last_beat;
91 | @(posedge inter.ACLK) disable iff (!inter.ARESETn)
92 | (inter.WVALID && inter.WREADY && inter.WLAST) |=> !inter.WREADY;
93 endproperty
94 A_WLAST_LAST_BEAT: assert property (wlast_on_last_beat)
95 | else $error("WREADY should deassert after WLAST");
96 C_WLAST_LAST_BEAT: cover property (wlast_on_last_beat);
97
98 // Write response must come after write data completion
99 property write_order;
100 | @(posedge inter.ACLK) disable iff (!inter.ARESETn)
101 | $rose(inter.BVALID) |-> $past(inter.WVALID && inter.WREADY && inter.WLAST);
102 endproperty
103 A_WRITE_ORDER_DATA_RESP: assert property (write_order)
104 | else $error("Write response cannot start without data completion");
105 C_WRITE_ORDER_DATA_RESP: cover property (write_order);

```

```

107 // BVALID should be asserted after write data completion
108 property bvalid_after_wlast;
109   @(posedge inter.ACLK) disable iff (!inter.ARESETn)
110   (inter.WVALID && inter.WREADY && inter.WLAST) | => inter.BVALID;
111 endproperty
112 A_BVALID_AFTER_WLAST: assert property (bvalid_after_wlast)
113   else $error("BVALID should be asserted after WLAST handshake");
114 C_BVALID_AFTER_WLAST: cover property (bvalid_after_wlast);
115
116 // BVALID should remain stable until BREADY
117 property bvalid_stable;
118   @(posedge inter.ACLK) disable iff (!inter.ARESETn)
119   inter.BVALID && !inter.BREADY | => inter.BVALID;
120 endproperty
121 A_BVALID_STABLE: assert property (bvalid_stable)
122   else $error("BVALID must remain stable until handshake");
123 C_BVALID_STABLE: cover property (bvalid_stable);
124
125 // BRESP should remain stable when BVALID is high
126 property bresp_stable;
127   @(posedge inter.ACLK) disable iff (!inter.ARESETn)
128   inter.BVALID && !inter.BREADY | => $stable(inter.BRESP);
129 endproperty
130 A_BRESP_STABLE: assert property (bresp_stable)
131   else $error("BRESP must remain stable when BVALID is high");
132 C_BRESP_STABLE: cover property (bresp_stable);
133
134 // BVALID should deassert after handshake
135 property bvalid_deassert;
136   @(posedge inter.ACLK) disable iff (!inter.ARESETn)
137   (inter.BVALID && inter.BREADY) | => !inter.BVALID;
138 endproperty
139 A_BVALID_DEASSERT: assert property (bvalid_deassert)
140   else $error("BVALID should deassert after response handshake");
141 C_BVALID_DEASSERT: cover property (bvalid_deassert);

```

```

143 // ARREADY should go low after accepting address
144 property arready_deassert;
145   @(posedge inter.ACLK) disable iff (!inter.ARESETn)
146   (inter.ARVALID && inter.ARREADY) | => !inter.ARREADY;
147 endproperty
148 A_ARREADY_DEASSERT: assert property (arready_deassert)
149   else $error("ARREADY should deassert after address handshake");
150 C_ARREADY_DEASSERT: cover property (arready_deassert);
151
152 // Read address signals should remain stable when ARVALID is high
153 property araddr_stable;
154   @(posedge inter.ACLK) disable iff (!inter.ARESETn)
155   inter.ARVALID && !inter.ARREADY | => $stable(inter.ARADDR);
156 endproperty
157 A_ARADDR_STABLE: assert property (araddr_stable)
158   else $error("ARADDR must remain stable when ARVALID is high");
159 C_ARADDR_STABLE: cover property (araddr_stable);
160
161 property arlen_stable;
162   @(posedge inter.ACLK) disable iff (!inter.ARESETn)
163   inter.ARVALID && !inter.ARREADY | => $stable(inter.ARLLEN);
164 endproperty
165 A_ARLEN_STABLE: assert property (arlen_stable)
166   else $error("ARLEN must remain stable when ARVALID is high");
167 C_ARLEN_STABLE: cover property (arlen_stable);
168
169 property arsize_stable;
170   @(posedge inter.ACLK) disable iff (!inter.ARESETn)
171   inter.ARVALID && !inter.ARREADY | => $stable(inter.ARSIZE);
172 endproperty
173 A_ARSIZE_STABLE: assert property (arsize_stable)
174   else $error("ARSIZE must remain stable when ARVALID is high");
175 C_ARSIZE_STABLE: cover property (arsize_stable);

```

```

178 // RVALID should be asserted after read address
179 property rvalid_after_araddr;
180   @(posedge inter.ACLK) disable iff (!inter.ARESETn)
181   (inter.ARVALID && inter.ARREADY) |-> ##[1:3] inter.RVALID;
182 endproperty
183 A_RVALID_AFTER_ARADDR: assert property (rvalid_after_araddr)
184   else $error("RVALID should be asserted within 3 cycles after read address");
185 C_RVALID_AFTER_ARADDR: cover property (rvalid_after_araddr);
186
187 // RVALID should remain stable until RREADY
188 property rvalid_stable;
189   @(posedge inter.ACLK) disable iff (!inter.ARESETn)
190   inter.RVALID && !inter.RREADY |=> inter.RVALID;
191 endproperty
192 A_RVALID_STABLE: assert property (rvalid_stable)
193   else $error("RVALID must remain stable until handshake");
194 C_RVALID_STABLE: cover property (rvalid_stable);
195
196 // RRESP should remain stable when RVALID is high
197 property rresp_stable;
198   @(posedge inter.ACLK) disable iff (!inter.ARESETn)
199   inter.RVALID && !inter.RREADY |=> $stable(inter.RRESP);
200 endproperty
201 A_RRESP_STABLE: assert property (rresp_stable)
202   else $error("RRESP must remain stable when RVALID is high");
203 C_RRESP_STABLE: cover property (rresp_stable);
204
205 // RLAST should remain stable when RVALID is high
206 property rlast_stable;
207   @(posedge inter.ACLK) disable iff (!inter.ARESETn)
208   inter.RVALID && !inter.RREADY |=> $stable(inter.RLAST);
209 endproperty
210 A_RLAST_STABLE: assert property (rlast_stable)
211   else $error("RLAST must remain stable when RVALID is high");
212 C_RLAST_STABLE: cover property (rlast_stable);

```

```

214 // BRESP should be OKAY (00) or SLVERR (10)
215 property bresp_valid_values;
216   @(posedge inter.ACLK) disable iff (!inter.ARESETn)
217   inter.BVALID |-> (inter.BRESP == 2'b00 || inter.BRESP == 2'b10);
218 endproperty
219 A_BRESP_VALID_VALUES: assert property (bresp_valid_values)
220   else $error("Invalid BRESP value: %b", inter.BRESP);
221 C_BRESP_VALID_VALUES: cover property (bresp_valid_values);
222
223 // RRESP should be OKAY (00) or SLVERR (10)
224 property rresp_valid_values;
225   @(posedge inter.ACLK) disable iff (!inter.ARESETn)
226   inter.RVALID |-> (inter.RRESP == 2'b00 || inter.RRESP == 2'b10);
227 endproperty
228 A_RRESP_VALID_VALUES: assert property (rresp_valid_values)
229   else $error("Invalid RRESP value: %b", inter.RRESP);
230 C_RRESP_VALID_VALUES: cover property (rresp_valid_values);
231
232 // 4KB boundary crossing should result in SLVERR for write
233 property write_boundary;
234   @(posedge inter.ACLK) disable iff (!inter.ARESETn)
235   (inter.BVALID &&
236    (((inter.AWADDR & 16'h0FFF) + ((inter.AWLEN + 1) << inter.AWSIZE)) > 16'h0FFF))
237   |-> inter.BRESP == 2'b10;
238 endproperty
239 A_WRITE_BOUNDARY_ERROR: assert property (write_boundary)
240   else $error("4KB boundary crossing should result in SLVERR");
241 C_WRITE_BOUNDARY_ERROR: cover property (write_boundary);

```

```

243 // Out of memory range should result in SLVERR for write
244 property write_range;
245   @(posedge inter.ACLK) disable iff (!inter.ARESETn)
246   (inter.BVALID && ((inter.AWADDR >> 2) >= 1024))
247   |-> inter.BRESP == 2'b10;
248 endproperty
249 A_WRITE_RANGE_ERROR: assert property (write_range)
250   else $error("Out of range write should result in SLVERR");
251 C_WRITE_RANGE_ERROR: cover property (write_range);
252
253 // 4KB boundary crossing should result in SLVERR for read
254 property read_boundary;
255   @(posedge inter.ACLK) disable iff (!inter.ARESETn)
256   (inter.RVALID &&
257    (((inter.ARADDR & 16'h0FFF) + ((inter.ARLen + 1) << inter.ARSIZE)) > 16'h0FFF))
258   |-> inter.RRESP == 2'b10;
259 endproperty
260 A_READ_BOUNDARY_ERROR: assert property (read_boundary)
261   else $error("4KB boundary crossing should result in SLVERR for read");
262 C_READ_BOUNDARY_ERROR: cover property (read_boundary);
263
264 // Out of memory range should result in SLVERR for read
265 property read_range;
266   @(posedge inter.ACLK) disable iff (!inter.ARESETn)
267   (inter.RVALID && ((inter.ARADDR >> 2) >= 1024))
268   |-> inter.RRESP == 2'b10;
269 endproperty
270 A_READ_RANGE_ERROR: assert property (read_range)
271   else $error("Out of range read should result in SLVERR");
272 C_READ_RANGE_ERROR: cover property (read_range);
273
274 endmodule

```

## IV. System Results

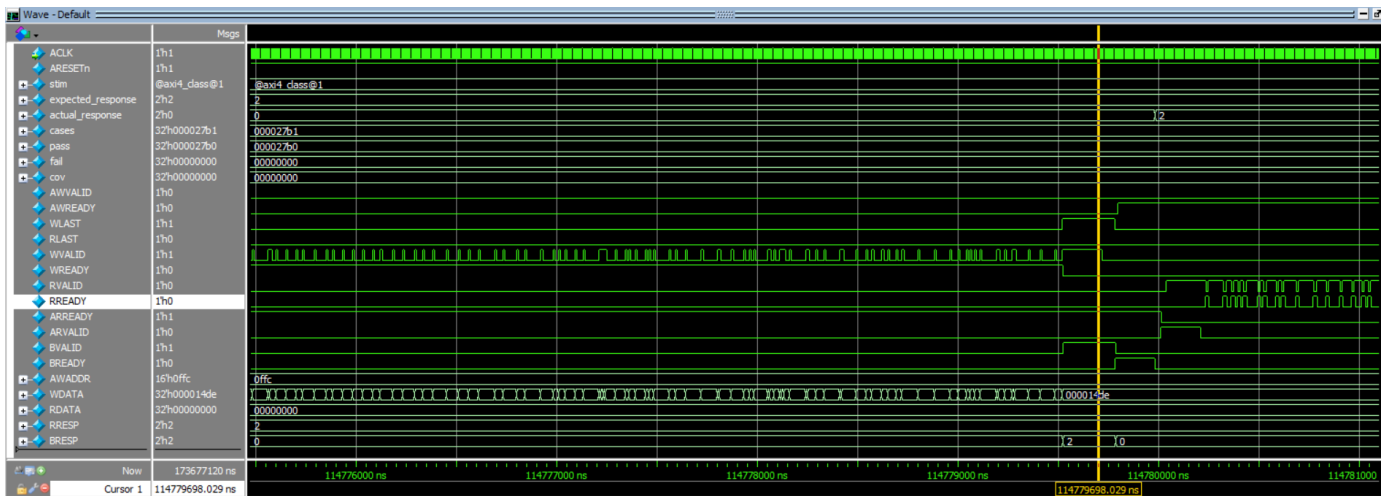
```

# TIME: $0t          889570840
# Test 163199
# Write Test
# ADDR = 4092, LEN = 128, SIZE = 2, Memory Access = 1152
# Delays - AW:7, W:0, B:6, AR:0, R:3
# Test 163199 PASSED
# write Operation passed
#
# TIME: $0t          889571520
# Test 163200
# Write Test
# ADDR = 4092, LEN = 0, SIZE = 2, Memory Access = 1024
# Delays - AW:1, W:5, B:1, AR:7, R:0
# Test 163200 PASSED
# write Operation passed
#
# === Final Results ===
# Number of tests: 163200, Passed: 163200, Failed: 0
# Coverage achieved: 100.0%
# ** Note: $stop      : Axi4_test.sv(79)
#      Time: 889572180 ns Iteration: 0 Instance: /top/tb
# Break in Module Axi4_test at Axi4_test.sv line 79

```



Cover Directives														
Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
/top/check/C_RESE... SVA		✓	Off	1	1	Unli...	1	100%		✓	0	80	15000 ps	1
/top/check/C_RESE... SVA		✓	Off	1	1	Unli...	1	100%		✓	0	80	15000 ps	1
/top/check/C_RESE... SVA		✓	Off	1	1	Unli...	1	100%		✓	0	80	15000 ps	1
/top/check/C_RESE... SVA		✓	Off	1	1	Unli...	1	100%		✓	0	80	15000 ps	1
/top/check/C_RESE... SVA		✓	Off	1	1	Unli...	1	100%		✓	0	80	15000 ps	1
/top/check/C_RESE... SVA		✓	Off	1	1	Unli...	1	100%		✓	0	80	15000 ps	1
/top/check/C_AWR... SVA		✓	Off	81528	1	Unli...	1	100%		✓	0	80	26065000 ps	81528
/top/check/C_AWA... SVA		✓	Off	...032	1	Unli...	1	100%		✓	0	160	26085000 ps	1549032
/top/check/C_AWL... SVA		✓	Off	...032	1	Unli...	1	100%		✓	0	160	26085000 ps	1549032
/top/check/C_AWS... SVA		✓	Off	...032	1	Unli...	1	100%		✓	0	160	26085000 ps	1549032
/top/check/C_WDA... SVA		✓	Off	...032	1	Unli...	1	100%		✓	0	160	33545000 ps	1549032
/top/check/C_WLA... SVA		✓	Off	81528	1	Unli...	1	100%		✓	0	80	33525000 ps	81528
/top/check/C_WRI... SVA		✓	Off	81528	1	Unli...	1	100%		✓	0	80	33535000 ps	81528
/top/check/C_BVAL... SVA		✓	Off	81528	1	Unli...	1	100%		✓	0	80	33525000 ps	81528
/top/check/C_BVAL... SVA		✓	Off	...592	1	Unli...	1	100%		✓	0	160	33545000 ps	1976592
/top/check/C_BRES... SVA		✓	Off	...592	1	Unli...	1	100%		✓	0	160	33545000 ps	1976592
/top/check/C_BVAL... SVA		✓	Off	81528	1	Unli...	1	100%		✓	0	80	33765000 ps	81528
/top/check/C_ARA... SVA		✓	Off	81672	1	Unli...	1	100%		✓	0	80	55000 ps	81672
/top/check/C_ARA... SVA		✓	Off	...768	1	Unli...	1	100%		✓	0	160	75000 ps	1551768
/top/check/C_ARLE... SVA		✓	Off	...768	1	Unli...	1	100%		✓	0	160	75000 ps	1551768
/top/check/C_ARSL... SVA		✓	Off	...768	1	Unli...	1	100%		✓	0	160	75000 ps	1551768
/top/check/C_RVAL... SVA		✓	Off	81672	1	Unli...	1	100%		✓	0	80	55000 ps	81672
/top/check/C_RVAL... SVA		✓	Off	...583	1	Unli...	1	100%		✓	0	160	95000 ps	37606583
/top/check/C_RRE... SVA		✓	Off	...583	1	Unli...	1	100%		✓	0	160	95000 ps	37606583
/top/check/C_RLAS... SVA		✓	Off	...583	1	Unli...	1	100%		✓	0	160	95000 ps	37606583
/top/check/C_BRES... SVA		✓	Off	...120	1	Unli...	1	100%		✓	0	80	33535000 ps	2058120
/top/check/C_RESE... SVA		✓	Off	...295	1	Unli...	1	100%		✓	0	80	85000 ps	47954295
/top/check/C_WRI... SVA		✓	Off	45168	1	Unli...	1	100%		✓	0	80	554518235000 ps	545168
/top/check/C_WRI... SVA		✓	Off	75361	1	Unli...	1	100%		✓	0	80	554500615000 ps	475361
/top/check/C_REA... SVA		✓	Off	...652	1	Unli...	1	100%		✓	0	80	554519425000 ps	14467652
/top/check/C_REA... SVA		✓	Off	...507	1	Unli...	1	100%		✓	0	80	554501825000 ps	12091507



## Function coverage

Covergroups										
Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment	Missing Bins
/interface_top_sv...		99.99%								
TYPB_axi4_cov...		99.99%	100	99.99%		✓	auto(1)			3
CVP_axi4_c...		100.00%	100	100.00%		✓				0
CVP_axi4_c...		100.00%	100	100.00%		✓				0
CVP_axi4_c...		100.00%	100	100.00%		✓				0
CVP_axi4_c...		100.00%	100	100.00%		✓				0
CVP_axi4_c...		100.00%	100	100.00%		✓				0
CVP_axi4_c...		100.00%	100	100.00%		✓				0
CVP_axi4_c...		100.00%	100	100.00%		✓				0
CVP_axi4_c...		100.00%	100	100.00%		✓				0
CROSS_axi...		100.00%	100	100.00%		✓				0
CROSS_axi...		99.93%	100	99.93%		✓				3
CROSS_axi...		100.00%	100	100.00%		✓				0
CROSS_axi...		100.00%	100	100.00%		✓				0
CROSS_axi...		100.00%	100	100.00%		✓				0



## Code coverage

```
FSM Coverage:
  Enabled Coverage          Bins      Hits      Misses  Coverage
  -----
  FSM States                7        7         0    100.00%
  FSM Transitions          10        7         3     70.00%

=====FSM Details=====

FSM Coverage for instance /top/dut --

FSM_ID: write_state
Current State Object : write_state
-----
State Value MapInfo :
-----
Line      State Name      Value
-----
104        W_IDLE          0
120        W_ADDR          1
125        W_DATA          2
147        W_RESP          3
Covered States :
-----
          State      Hit_count
          -----
          W_IDLE      21003
          W_ADDR      21000
          W_DATA      21000
          W_RESP      21000
Covered Transitions :
-----
Line      Trans_ID      Hit_count      Transition
-----
116        0            21000          W_IDLE -> W_ADDR
```

- **Comment:** Fsm Transitions has 70% coverage as it doesn't handle moving from data to addr or from resp to data or addr.

```
Statement Coverage:
  Enabled Coverage          Bins      Hits      Misses  Coverage
  -----
  Statements                83        83         0    100.00%

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

Statement Coverage for instance /top/dut --

Line      Item      Count      Source
-----
File axi4.sv
24         1         3
25         1         3
28         1        20457
29         1        20457
32         1       1905673
33         1       1905673
62         1      21422144
65         1         2
66         1         2
67         1         2
68         1         2
70         1         2
71         1         2
```

```

Expression Coverage:
  Enabled Coverage
  -----
  Expressions      Bins   Covered   Misses  Coverage
                   ----   -
                   5      5         0    100.00%

=====Expression Details=====

Expression Coverage for instance /top/dut --

  File axi4.sv
  -----Focused Expression View-----
Line      28 Item   1 (((inter.AWADDR & 4095) + ((inter.AWLEN + 1) << inter.AWSIZE)) > 4095)
Expression totals: 1 of 1 input term covered = 100.00%

  -----Focused Expression View-----
Line      29 Item   1 (((inter.ARADDR & 4095) + ((inter.ARLLEN + 1) << inter.ARSIZE)) > 4095)
Expression totals: 1 of 1 input term covered = 100.00%

  -----Focused Expression View-----
Line      32 Item   1 ((write_addr >> 2) < 1024)
Expression totals: 1 of 1 input term covered = 100.00%

  -----Focused Expression View-----
Line      33 Item   1 ((read_addr >> 2) < 1024)
Expression totals: 1 of 1 input term covered = 100.00%

  -----Focused Expression View-----
Line     196 Item   1 (read_burst_cnt == 0)
Expression totals: 1 of 1 input term covered = 100.00%

```

```

=====
=== Instance: /top/dut
=== Design Unit: work.axi4
=====

Branch Coverage:
  Enabled Coverage
  -----
  Branches      Bins   Hits   Misses  Coverage
                   ----   -
                   35    35         0    100.00%

=====Branch Details=====

Branch Coverage for instance /top/dut

  Line      Item      Count      Source
  ----      -
  File axi4.sv
  -----IF Branch-----
    63              21422144    Count coming in to IF
    63          1          2
    95          1    21422142
Branch totals: 2 hits of 2 branches = 100.00%

  -----CASE Branch-----
    103              21422142    Count coming in to CASE
    104          1    11875720
    120          1     21000
    125          1    8959065
    147          1    566356
    155          1         1
Branch totals: 5 hits of 5 branches = 100.00%

  -----IF Branch-----
    109              11875720    Count coming in to IF
    109          1     21000
                   11854720    All False Count
Branch totals: 2 hits of 2 branches = 100.00%

```

```

=====
Toggle Coverage:
  Enabled Coverage      Bins    Hits    Misses  Coverage
  -----
  Toggles               272     247      25    90.80%

=====Toggle Details=====

Toggle Coverage for instance /top/inter --

                                Node      1H->0L    0L->1H  "Coverage"
                                -----
                                ARADDR[1-0]      0         0      0.00
                                ARSIZE[2-0]        0         0      0.00
                                AWADDR[1-0]        0         0      0.00
                                AWSIZE[2-0]        0         0      0.00
                                BRESP[0]          0         0      0.00
                                RRESP[1]          0         1     50.00
                                RRESP[0]          0         0      0.00

Total Node Count    =      136
Toggled Node Count  =      123
Untoggled Node Count =       13

Toggle Coverage     =      90.80% (247 of 272 bins)

```

- **Comment:** Toggle 90% coverage as resp is always 0 or 2, so the first bit doesn't toggle, and size is always 2, so it doesn't toggle, and the first 2 bits of addr are always 0 also.

```

Condition Coverage:
  Enabled Coverage      Bins  Covered  Misses  Coverage
  -----
  Conditions            23     18       5    78.26%

=====Condition Details=====

Condition Coverage for instance /top/dut --

File axi4.sv
-----Focused Condition View-----
Line      109 Item      1 (inter.AWVALID && inter.AWREADY)
Condition totals: 1 of 2 input terms covered = 50.00%

  Input Term  Covered  Reason for no coverage  Hint
  -----
  inter.AWVALID      Y
  inter.AWREADY      N  '_0' not hit      Hit '_0'

  Rows:      Hits  FEC Target      Non-masking condition(s)
  -----
  Row  1:      1  inter.AWVALID_0      -
  Row  2:      1  inter.AWVALID_1      inter.AWREADY
  Row  3:  ***0***  inter.AWREADY_0      inter.AWVALID
  Row  4:      1  inter.AWREADY_1      inter.AWVALID

-----Focused Condition View-----
Line      126 Item      1 (inter.WVALID && inter.WREADY)
Condition totals: 1 of 2 input terms covered = 50.00%

  Input Term  Covered  Reason for no coverage  Hint
  -----
  inter.WVALID      Y
  inter.WREADY      N  '_0' not hit      Hit '_0'

  Rows:      Hits  FEC Target      Non-masking condition(s)
  -----

```

```

-----Focused Condition View-----
Line      109 Item    1  (inter.AWVALID && inter.AWREADY)
Condition totals: 1 of 2 input terms covered = 50.00%

  Input Term  Covered  Reason for no coverage  Hint
  -----
  inter.AWVALID      Y
  inter.AWREADY      N  '_0' not hit      Hit '_0'

  Rows:      Hits  FEC Target      Non-masking condition(s)
  -----
Row  1:      1  inter.AWVALID_0      -
Row  2:      1  inter.AWVALID_1      inter.AWREADY
Row  3:  ***0***  inter.AWREADY_0      inter.AWVALID
Row  4:      1  inter.AWREADY_1      inter.AWVALID

-----Focused Condition View-----
Line      126 Item    1  (inter.WVALID && inter.WREADY)
Condition totals: 1 of 2 input terms covered = 50.00%

  Input Term  Covered  Reason for no coverage  Hint
  -----
  inter.WVALID      Y
  inter.WREADY      N  '_0' not hit      Hit '_0'

  Rows:      Hits  FEC Target      Non-masking condition(s)
  -----
Row  1:      1  inter.WVALID_0      -
Row  2:      1  inter.WVALID_1      inter.WREADY
Row  3:  ***0***  inter.WREADY_0      inter.WVALID
Row  4:      1  inter.WREADY_1      inter.WVALID

-----Focused Condition View-----
Line      127 Item    1  (write_addr_valid && ~write_boundary_cross)
Condition totals: 2 of 2 input terms covered = 100.00%

-----Focused Condition View-----
Line      134 Item    1  (inter.WLAST || (write_burst_cnt == 0))
Condition totals: 1 of 2 input terms covered = 50.00%

```

- **Comment:** condition coverage has only 78.26% because we sample only input values, not output as AWREADY and WREADY, we don't change the values of these signals, which causes not all conditions to be hit.

# V. Run.do

```
1 vlib work
2 vlog *.sv
3 vsim -assertdebug +acc -voptargs=+acc work.top
4 do wave.do
5
6 run -all
7
```

```
1 vlib work
2 vlog *.*v +cover -covercells
3 vsim work.top -cover
4 coverage save -onexit cov.ucdb -du work.axi4
5 add wave -radix hex /top/dut/*
6 run -all
7 coverage report -details -output cov_report.txt
8
```