

AXI4 Verification Project

Abdelrahman Ahmed

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'hFFFFFFF]
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'hFFF_FFFF, 32'hAAAA_AAAA, 32'h5555_5555, 32'h1111_0000,
67                             32'h0000_1111}; // Fixed: removed invalid constant

```

```

70     // Coverage group
71     covergroup 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'hFFF_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]};

```

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         stim.addr_c.constraint_mode(1);
63         stim.addr_corner_c.constraint_mode(0);
64         stim.data_c1.constraint_mode(0);
65         stim.data_c2.constraint_mode(0);
66         stim.data_c3.constraint_mode(1);
67         stim.data_corner_c.constraint_mode(0);
68         repeat (10000) begin
69             cases++;
70             generate_stimulus();

```

```

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();

```

```

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

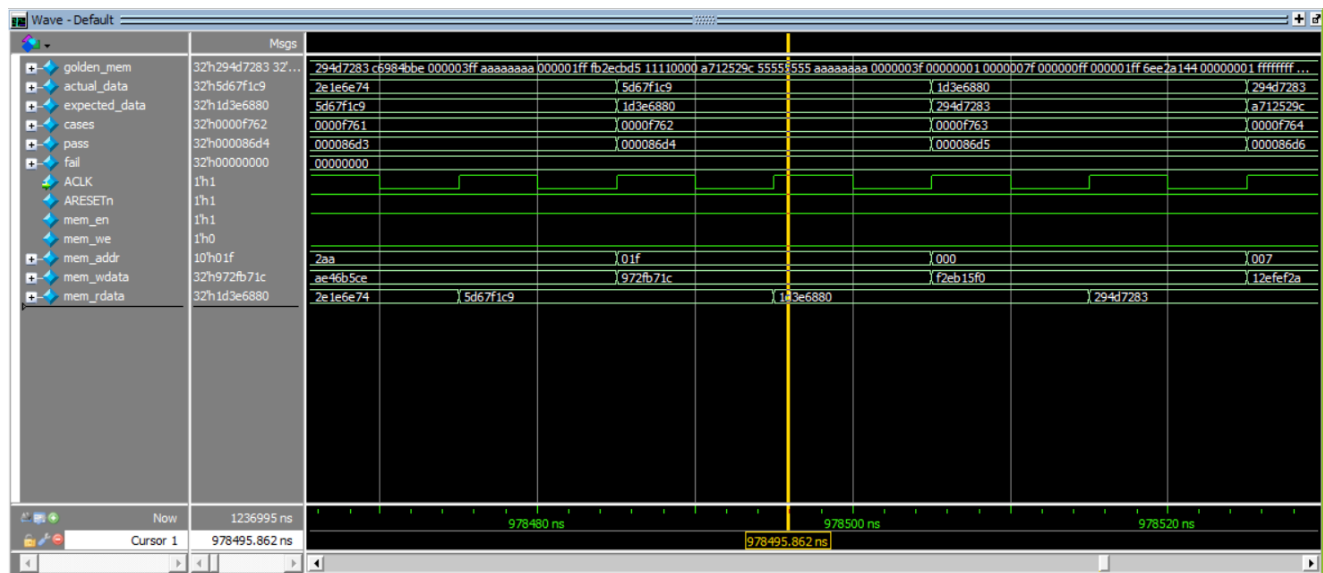
```

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%	100	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              2
  18              115704
  18              7993      All False Count
  Branch totals: 3 hits of 3 branches = 100.00%

  -----IF Branch-----
  19              115704      Count coming in to IF
  19              28333
  21              87371
  Branch totals: 3 hits of 3 branches = 100.00%

Statement Coverage:
  Enabled Coverage      Bins      Hits      Misses      Coverage
  -----
  Statements              7         7         0      100.00%

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

Statement Coverage for instance /top/dut --

```

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;
```

```

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,

```

```

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
34     ^axi4 ^inter ^
35     ^
36     ^
37     ^
38     ^
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.ALEN + 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
```



```

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 address corners
62 constraint data_c1 {
63     DATA inside {
64         [32'd0 : 32'd255],
65         [32'd256 : 32'd1023],
66         [32'd1024 : 32'd4095],
67         [32'd4096 : 32'd16383],
68         [32'd16384 : 32'd32767]
69     };
70 }
71
72
73
74
75
76 constraint data_c2 {

```

```

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 // Fixed size coverage
130 coverpoint SIZE {
131     bins fixed_size = {2};
132     illegal_bins others = default;
133 }
134
135 // DATA coverage with corner bins
136 coverpoint DATA {
137     bins corner0 = {32'd0};
138     bins corner1 = {32'd1};
139     bins corner2 = {32'hFFFF_FFFF};

```

```

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.

```

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

```

```

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

```

```
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

```

```

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++)

```

```

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

```

```

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 ..
263 task collect();
264     actual_queue.delete();
265     // Start with RREADY high, then apply delay by temporarily deasserting
266     inter.RREADY = 1'b1;
267
268     for (int i = 0; i <= stim.LEN; i++)
269     begin
270         // Apply r_ready_delay for each read beat
271         if (stim.r_ready_delay > 0)
272         begin
273             inter.RREADY = 1'b0;

```



```

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     end else
320     if (stim.OPERATION == 2'd2)
321     begin
322         $display("Test %0d", cases);
323         $display("Read Test");
324         stim.display();
325     end
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         end else
335         begin
336             fail++;
337             $display("Test %0d FAILED Resopnse mismatch", cases);
338             $display("Read Operation failed");
339         end
340     end
341 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     end

```

```

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

```

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;
```

```

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 // BVALID should be asserted after write data completion
106 property bvalid_after_wlast;
107     @(posedge inter.ACLK) disable iff (!inter.ARESETn)
108     (inter.WVALID && inter.WREADY && inter.WLAST) | => inter.BVALID;
109 endproperty
110 A_BVALID_AFTER_WLAST: assert property (bvalid_after_wlast)
111     else $error("BVALID should be asserted after WLAST handshake");
112 C_BVALID_AFTER_WLAST: cover property (bvalid_after_wlast);
113
114 // BVALID should remain stable until BREADY
115 property bvalid_stable;
116     @(posedge inter.ACLK) disable iff (!inter.ARESETn)
117     inter.BVALID && !inter.BREADY | => inter.BVALID;
118 endproperty

```

```

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.ARLen);
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
176 // RVALID should be asserted after read address
177 property rvalid_after_araddr;
178   @(posedge inter.ACLK) disable iff (!inter.ARESETn)
179     (inter.ARVALID && inter.ARREADY) | -> ##[1:3] inter.RVALID;
180 endproperty
181 A_RVALID_AFTER_ARADDR: assert property (rvalid_after_araddr)
182   else $error("RVALID should be asserted within 3 cycles after read address");
183 C_RVALID_AFTER_ARADDR: cover property (rvalid_after_araddr);
184
185 // RVALID should remain stable until RREADY
186 property rvalid_stable;
187   @(posedge inter.ACLK) disable iff (!inter.ARESETn)

```

```

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);

```













































































```

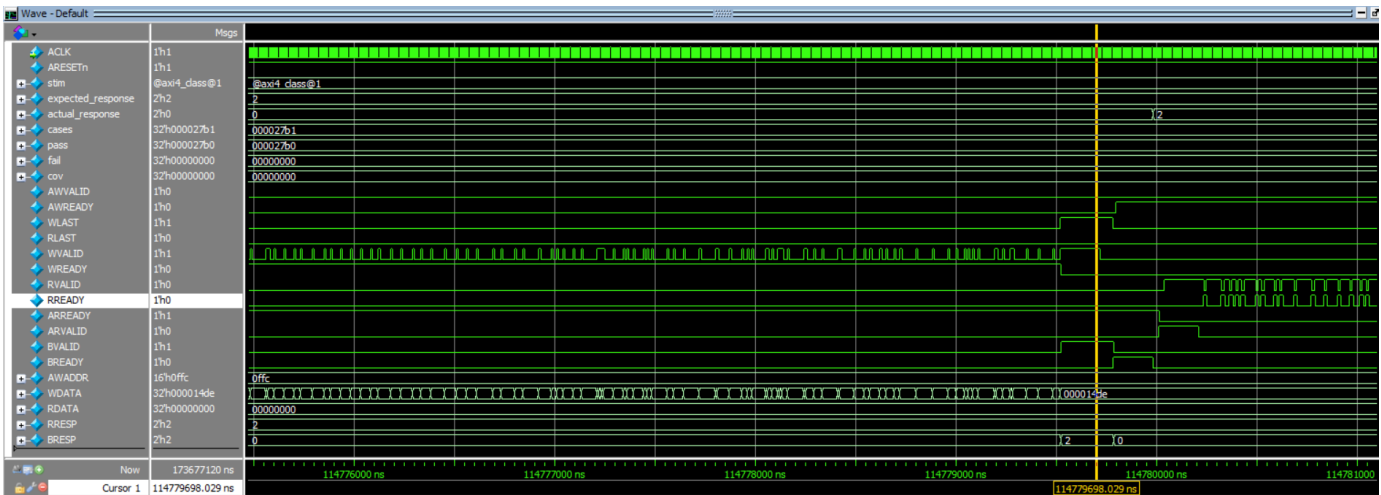
243 // Out of memory range should result in SLVERR for write
244 property write_range;
245   @(posedge inter.ACLK) disable iff (!inter.ARESETn)
246   (inter.BVALID && ((inter.AWADDR >> 2) >= 1024))
247   |-> inter.BRESP == 2'b10;
248 endproperty
249 A_WRITE_RANGE_ERROR: assert property (write_range)
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 &&

```

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_ARR... SVA			Off	81672	1	Unli...	1	100%			0	80	55000 ps	81672
 /top/check/C_AWA... SVA			Off	760	1	Unli...	1	100%			0	160	25000 ps	1551200



Function coverage

Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_Instances	Get_inst_coverage	Comment	Missing Bins
/interface_top_sv_...		99.99%								
TYPE 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

```

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 --

```


- **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

Expression Coverage:
  Enabled Coverage      Bins  Covered  Misses  Coverage
  -----
  Expressions           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.ARLEN + 1) << inter.ARSIZE)) > 4095)
```

```

=====
=== Instance: /top/dut
=== Design Unit: work.axi4
=====
Branch Coverage:
  Enabled Coverage      Bins      Hits      Misses  Coverage
  -----
  Branches              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

```

- **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
  -----
  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      -
```

- **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
```