AXI4 Verification Project

Abdelrahman Ahmed

I. Memory

1. Interface

```
interface memory_interface (input bit ACLK);

    //memory signals

    logic           ARESETn;
    logic           mem_en;
    logic           mem_we;
    logic [9:0]     mem_addr;
    logic [31:0]    mem_wdata;
    logic [31:0]    mem_rdata;

    modport dut (
        input   ACLK,
        input   ARESETn,
        input   mem_en,
        input   mem_we,
        input   mem_addr,
        input   mem_wdata,
        output  mem_rdata
    );

    modport tb (
        input   ACLK,
        output  ARESETn,
        output  mem_en,
        output  mem_we,
        output  mem_addr,
        output  mem_wdata,
        input   mem_rdata
    );


endinterface
```

```
`include "memory_interface.sv"
`include "mem_test.sv"
`include "axi_memory.sv"
module top();

  //memory signals

  logic           mem_en;
  logic           mem_we;
  logic [9:0]     mem_addr;
  logic [31:0]    mem_wdata;
  logic [31:0]    mem_rdata;

  logic ACLK;
  initial begin
  ACLK = 0;
  forever begin
    #5ns ACLK = ~ACLK;
  end
  end

  memory_interface  inter  (ACLK);
  axi4_memory       dut    (inter.dut);
  mem_test          tb     (inter.tb);
  mem_assert        check  (inter.dut);

endmodule
```

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

```
class memory_class;
    // Memory signals
    rand logic        mem_en;
    rand logic        mem_we;
    rand logic [9:0]  mem_addr;
    rand logic [31:0] mem_wdata;

    // Constraint for memory enable
    constraint mem_en_const {
        mem_en dist {0 := 1, 1 := 9};
    }

    // Constraint for memory write enable
    constraint mem_we_const {
        if (mem_en)
            mem_we dist {0 := 6, 1 := 4};  // 0 = read, 1 = write
        else
            mem_we == 0;
    }

    // Address distribution constraint
    constraint addr_c {
        mem_addr inside {[10'd0 : 10'd1023]};
        };

    // Address corner case constraint
    constraint addr_corner_c {
        mem_addr inside {10'd0, 10'd1, 10'd7, 10'd15, 10'd31, 10'd63, 10'd127, 10'd255, 10'd511,
                         10'd1023, 10'b1010101010, 10'b0101010101, 10'b1111111111,10'b1111100000,
                         10'b0000011111};
    }

```

```
constraint data_c1 {
  mem_wdata inside {
    [32'd0          : 32'd255],
    [32'd256        : 32'd1023],
    [32'd1024       : 32'd4095],
    [32'd4096       : 32'd16383],
    [32'd16384      : 32'd32767]
  };
}

constraint data_c2 {
  mem_wdata inside {
    [32'b0          : 32'b1],
    [32'd32768      : 32'd49151],
    [32'd49152      : 32'd65535],
    [32'd65536      : 32'd262143],
    [32'd262144     : 32'd1048575]
  };
}

constraint data_c3 {
  mem_wdata inside {
    [32'b0          : 32'b1],
    [32'd1048576    : 32'd16777215],
    [32'd16777216   : 32'd268435455],
    [32'd268435456  : 32'd1073741823],
    [32'd1073741824 : 32'hFFFFFFFF]
  };
}

  constraint data_corner_c {
    mem_wdata inside {32'd0, 32'd1, 32'd7, 32'd15, 32'd31, 32'd63, 32'd127, 32'd255, 32'd511,
                      32'd1023, 32'hFFFF_FFFF, 32'hAAAA_AAAA, 32'h5555_5555, 32'h1111_0000,
                      32'h0000_1111}; // Fixed: removed invalid constant
```

```
    // Coverage group
covergroup Memory_cov;
    option.per_instance = 1;

    mem_en_cp : coverpoint mem_en {
        bins disabled = {0};
        bins enabled  = {1};
    }

    mem_we_cp : coverpoint mem_we iff (mem_en) {
        bins read_op  = {0};
        bins write_op = {1};
    }

    mem_addr_cp : coverpoint mem_addr iff (mem_en) {
        bins corners[] = {10'd0, 10'd1, 10'd7, 10'd15, 10'd31, 10'd63, 10'd127, 10'd255,
                          10'd511, 10'd1023, 10'b1010101010, 10'b0101010101, 10'b1111111111,
                          10'b1111100000, 10'b0000011111};

        bins all_values[] = {[10'd0 : 10'd1023]};
    }
```

```
    mem_wdata_cp : coverpoint mem_wdata iff (mem_en && mem_we) {
        bins corners[] = {32'd0, 32'd1, 32'd7, 32'd15, 32'd31, 32'd63, 32'd127, 32'd255,
                          32'd511, 32'd1023, 32'hFFFF_FFFF, 32'hAAAA_AAAA, 32'h5555_5555,
                          32'h1111_0000, 32'h0000_1111};

        bins grp1_1 = {[32'd0          : 32'd255]};
        bins grp1_2 = {[32'd256        : 32'd1023]};
        bins grp1_3 = {[32'd1024       : 32'd4095]};
        bins grp1_4 = {[32'd4096       : 32'd16383]};
        bins grp1_5 = {[32'd16384      : 32'd32767]};
```

4. Test

```
`timescale 1ns/1ps
`include "memory_class.sv"

module mem_test (memory_interface.tb inter);
  memory_class stim;
  logic [31:0] golden_mem [0:1023];
  logic [31:0] actual_data;
  logic [31:0] expected_data;
  int cases = 0;
  int pass  = 0;
  int fail  = 0;
  int cov   = 0;

  initial begin
    // Initialize interface signals
    inter.mem_en     = 0;
    inter.mem_we     = 0;
    inter.mem_addr   = 0;
    inter.mem_wdata  = 0;
    inter.ARESETn    = 0;
    cases            = 0;

    // Create stimulus object
    stim = new();

    // Initialize golden memory
    for (int i = 0; i < 1024; i++)
      golden_mem[i] = 0;

    // Reset sequence
    repeat (2) @(posedge inter.ACLK);
    inter.ARESETn = 1;
    repeat (2) @(posedge inter.ACLK); // Allow reset to complete
```

```
    stim.addr_c.constraint_mode(1);
    stim.addr_corner_c.constraint_mode(0);
    stim.data_c1.constraint_mode(1);
    stim.data_c2.constraint_mode(0);
    stim.data_c3.constraint_mode(0);
    stim.data_corner_c.constraint_mode(0);
    repeat (10000) begin
    cases++;
    generate_stimulus();
    golden_model();
    drive();
    collect();
    check();
    end

    stim.addr_c.constraint_mode(1);
    stim.addr_corner_c.constraint_mode(0);
    stim.data_c1.constraint_mode(0);
    stim.data_c2.constraint_mode(1);
    stim.data_c3.constraint_mode(0);
    stim.data_corner_c.constraint_mode(0);
    repeat (10000) begin
    cases++;
    generate_stimulus();
    golden_model();
    drive();
    stim.addr_c.constraint_mode(1);
    stim.addr_corner_c.constraint_mode(0);
    stim.data_c1.constraint_mode(0);
    stim.data_c2.constraint_mode(0);
    stim.data_c3.constraint_mode(1);
    stim.data_corner_c.constraint_mode(0);
    repeat (10000) begin
    cases++;
    generate_stimulus();
```

```
stim.addr_c.constraint_mode(0);
stim.addr_corner_c.constraint_mode(1);
stim.data_c1.constraint_mode(1);
stim.data_c2.constraint_mode(0);
stim.data_c3.constraint_mode(0);
stim.data_corner_c.constraint_mode(0);
repeat (10000) begin
cases++;
generate_stimulus();
golden_model();
drive();
collect();
check();
end

stim.addr_c.constraint_mode(0);
stim.addr_corner_c.constraint_mode(1);
stim.data_c1.constraint_mode(0);
stim.data_c2.constraint_mode(1);
stim.data_c3.constraint_mode(0);
stim.data_corner_c.constraint_mode(0);
repeat (10000) begin
cases++;
generate_stimulus();
golden_model();
drive();
collect();
check();
end
```

```
stim.addr_c.constraint_mode(0);
stim.addr_corner_c.constraint_mode(1);
stim.data_c1.constraint_mode(0);
stim.data_c2.constraint_mode(0);
stim.data_c3.constraint_mode(1);
stim.data_corner_c.constraint_mode(0);
repeat (10000) begin
cases++;
generate_stimulus();
golden_model();
```

```
    #200;
    $finish;
  end

  task generate_stimulus();
    if (!stim.randomize()) begin
      $display("ERROR: Randomization failed at test case %0d", cases);
      $finish;
    end
  endtask

  task golden_model();
    if (stim.mem_en) begin
      if (stim.mem_we) begin
        // Write operation
        golden_mem[stim.mem_addr] = stim.mem_wdata;
        expected_data = 'x; // Don't expect read data on write
      end else begin
        // Read operation
        expected_data = golden_mem[stim.mem_addr];
      end
    end else begin
      // Memory disabled
      expected_data = 'x;
    end
  endtask
```

```
  task drive();
    // Drive stimulus to interface
    inter.mem_en    = stim.mem_en;
    inter.mem_we    = stim.mem_we;
    inter.mem_addr  = stim.mem_addr;
    inter.mem_wdata = stim.mem_wdata;

    // Sample coverage for stimulus
    stim.Memory_cov.sample();

    @(posedge inter.ACLK);
  endtask

  task collect();

    if (stim.mem_en && !stim.mem_we)
    begin
      @(posedge inter.ACLK);
      actual_data = inter.mem_rdata;
    end else
    begin
      actual_data = 'x;
    end
  endtask
```

```
  task check();
    if (stim.mem_en && !stim.mem_we)
    begin
      if (actual_data !== expected_data)
      begin
        $display("FAIL [%0d]: Addr=0x%03x, Expected=0x%08x, Got=0x%08x",
                 cases, stim.mem_addr, expected_data, actual_data);
        fail++;
      end else
      begin
        pass++;
          $display("PASS [%0d]: Addr=0x%03x, Data=0x%08x",
                   cases, stim.mem_addr, actual_data);
      end
    end else if (stim.mem_en && stim.mem_we)
    begin
        $display("WRITE [%0d]: Addr=0x%03x, Data=0x%08x",
                 cases, stim.mem_addr, stim.mem_wdata);
    end
    $display("");
  endtask
endmodule
```

5. Assertions

```
module mem_assert (memory_interface.dut inter);

    // Property 1: Write operations
    property Write_prop;
        @(posedge inter.ACLK)
        disable iff (!inter.ARESETn)
        inter.mem_we |-> inter.mem_en;  // If mem_we is 1 , mem_en is must be 1
    endproperty

    A1: assert property (Write_prop)
        else $error("Write operation failed");
    C1: cover property (Write_prop);

    // Property 2: Read operation
    property Read_prop;
        @(posedge inter.ACLK)
        disable iff (!inter.ARESETn)
        (inter.mem_en && !inter.mem_we) |=> !$isunknown(inter.mem_rdata);
    endproperty

    A2: assert property (Read_prop)
        else $error("Read operation failed");
    C2: cover property (Read_prop);

    // Property 3: Write enable
    property enable_check;
        @(posedge inter.ACLK)
        disable iff (!inter.ARESETn)
        !inter.mem_en |-> !inter.mem_we;
    endproperty

    A3: assert property (enable_check)
        else $error("mem_we asserted when mem_en is low");
    C3: cover property (enable_check);

    // Property 4: Reset
    property reset_prop;
        @(posedge inter.ACLK)
        !inter.ARESETn |=> (inter.mem_rdata == 0);
    endproperty

    A4: assert property (reset_prop)
        else $error("Reset protocol failed");
    C4: cover property (reset_prop);

    // Property 5: Memory operations
    property stable_signals;
        @(posedge inter.ACLK)
        disable iff (!inter.ARESETn)
        inter.mem_en |-> (inter.mem_we == 1'b0 || inter.mem_we == 1'b1);
    endproperty

    A5: assert property (stable_signals)
        else $error("Invalid control signal values");
    C5: cover property (stable_signals);

    // Property 6: Address should be within valid range
    property valid_address;
        @(posedge inter.ACLK)
        disable iff (!inter.ARESETn)
        inter.mem_en |-> (inter.mem_addr < 1024);
    endproperty

    A6: assert property (valid_address)
        else $error("Invalid address");
    C6: cover property (valid_address);

endmodule
```

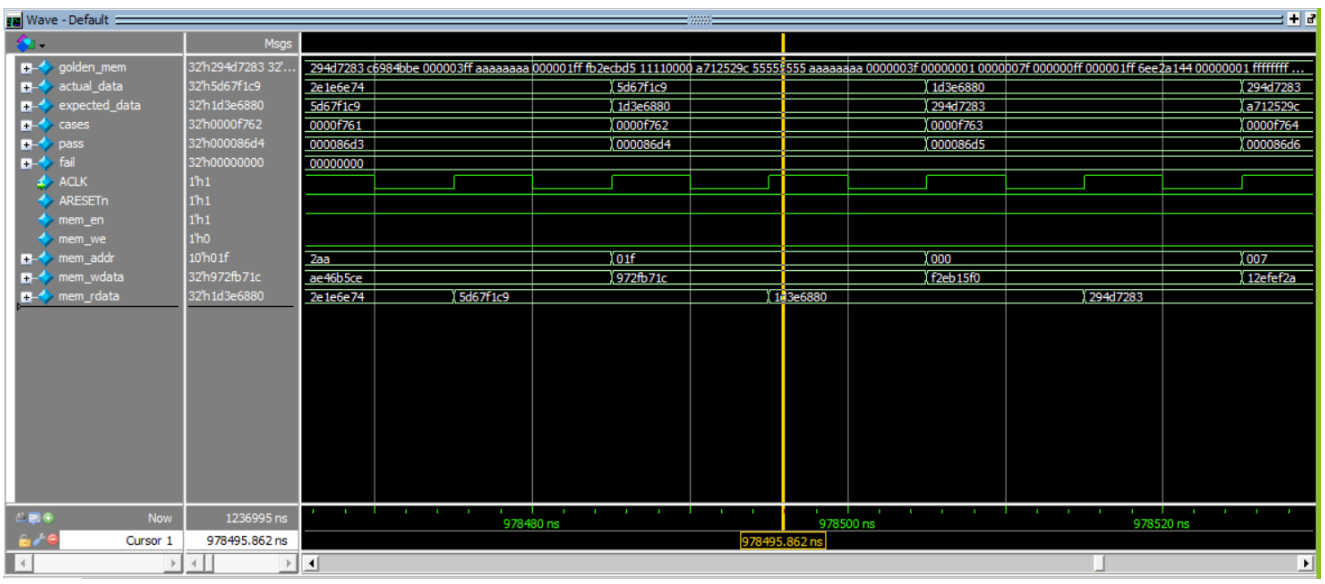
II. Memory Results

```
#
# === 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
```

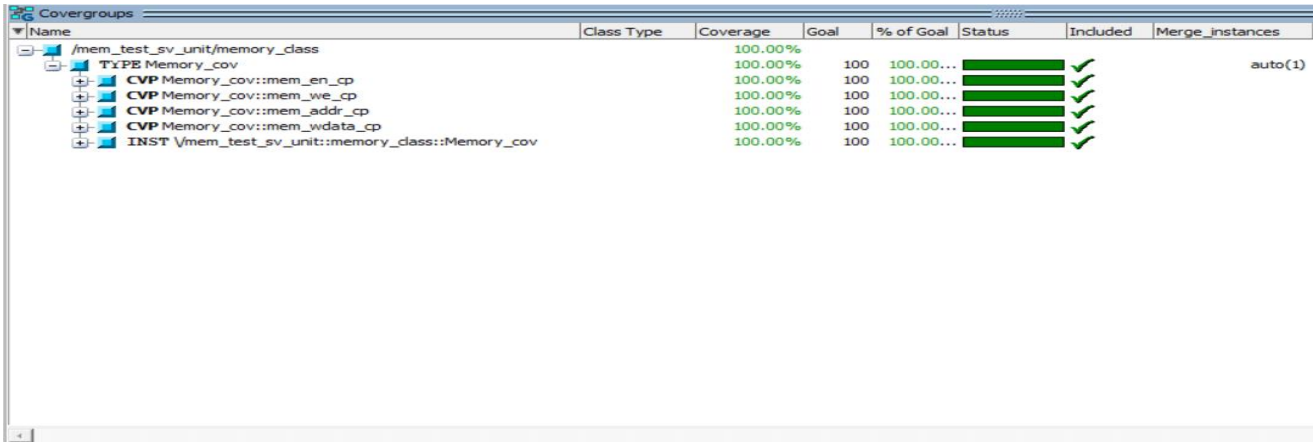
- **Comment**: Passed cases represent the cases that, when mem_en is asserted and the operation is read.

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



Covergroups

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 \/mem_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              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                       7         7         0   100.00%

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

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

III. System

1. Interface

```
interface axi4_interface (input bit ACLK);
    // axi4 signals
    bit             ARESETn;
    logic           AWVALID;
    logic           AWREADY;
    logic           WLAST;
    logic           RLAST;
    logic           WVALID;
    logic           WREADY;
    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,
```

```
`include "axi4_interface.sv"
`include "Axi4_test.sv"
`include "axi4.sv"
module top();

  // axi4 signals
  bit             ARESETn;
  logic           AWVALID;
  logic           AWREADY;
  logic           WLAST;
  logic           RLAST;
  logic           WVALID;
  logic           WREADY;
  logic           RVALID;
  logic           RREADY;
  logic           ARREADY;
  logic           ARVALID;
  logic           BVALID;
  logic           BREADY;
  logic [15:0]    AWADDR;
  logic [7:0]     AWLEN;
  logic [2:0]     AWSIZE;
  logic [31:0]    WDATA;
  logic [31:0]    ARADDR;
  logic [31:0]    RDATA;
  logic [7:0]     ARLEN;
  logic [2:0]     ARSIZE;
  logic [1:0]     RRESP;
  logic [1:0]     BRESP;

  bit ACLK ;
  initial begin
  ACLK = 0;
  axi4_interface inter (ACLK);
  axi4           dut   (inter.dut);
  Axi4_test      tb    (inter.tb);
  axi4_assert    check (inter.dut);

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

```
class axi4_class;

  // Write signals
  rand logic [15:0] ADDR;
  rand logic [7:0]  LEN;
  rand logic [2:0]  SIZE;
  rand logic [31:0] DATA;
  rand logic [1:0]  OPERATION;

  // Random delays for handshaking
```

```
  // Valid access constraint
  constraint valid_range {
    ((ADDR >> 2) + (LEN + 1)) < 1024;
  }

  // Invalid access constraint s
  constraint invalid_access_c {
    ((ADDR >> 2) + (LEN + 1)) >= 1024;
  }

  constraint LEN_range_c {
    LEN inside {[8'd0 : 8'd255]};
  }

  constraint LEN_corners_c {
    LEN inside {8'd0, 8'd1, 8'd127, 8'd128, 8'd254, 8'd255};
  }

  constraint addr_c {
    ADDR inside {[16'd0 : 16'd65535]};
  }

  constraint addr_corner_c {
constraint data_c1 {
  DATA inside {
    [32'd0          : 32'd255],
    [32'd256        : 32'd1023],
    [32'd1024       : 32'd4095],
    [32'd4096       : 32'd16383],
    [32'd16384      : 32'd32767]
  };
}

constraint data_c2 {
```

```
// Coverage
covergroup axi4_cov;

  // LEN coverage with corner bins
  coverpoint LEN {
    bins corner0   = {8'd0};
    bins corner1   = {8'd1};
    bins corner2   = {8'd127};
    bins corner3   = {8'd128};
    bins corner4   = {8'd254};
    bins corner5   = {8'd255};
    bins auto_bins[] = {[8'd0:8'd255]};
  }

  // ADDR coverage with corner bins
  coverpoint ADDR {
    bins corner0   = {16'd0};
    bins corner1   = {16'd1024};
    bins corner2   = {16'd2048};
    bins corner3   = {16'd4092};
    bins corner4   = {16'b1111_1111_1100};
    bins range_0   = { [16'd0       : 16'd255]   };
    bins range_1   = { [16'd256     : 16'd1023]  };
    bins range_2   = { [16'd1024    : 16'd4095]  };
    bins range_3   = { [16'd4096    : 16'd16383] };
    bins range_4   = { [16'd16384   : 16'd32767] };
    bins range_5   = { [16'd32768   : 16'd49151] };
    bins range_6   = { [16'd49152   : 16'd65535] };
  // Fixed size coverage
  coverpoint SIZE {
    bins fixed_size = {2};
    illegal_bins others = default;
  }

  // DATA coverage with corner bins
  coverpoint DATA {
    bins corner0       = {32'd0};
    bins corner1       = {32'd1};
    bins corner2       = {32'hFFFF_FFFF};
```

4.

```
  // Delay coverage
  coverpoint aw_valid_delay { bins all_values[] = {[0:7]}; }
  coverpoint w_valid_delay  { bins all_values[] = {[0:7]}; }
  coverpoint b_ready_delay  { bins all_values[] = {[0:7]}; }
  coverpoint ar_valid_delay { bins all_values[] = {[0:7]}; }
  coverpoint r_ready_delay  { bins all_values[] = {[0:7]}; }

  // Cross coverage
  cross LEN, ADDR;
  cross DATA, LEN;
  cross DATA, ADDR;
  cross aw_valid_delay, w_valid_delay;
  cross ar_valid_delay, r_ready_delay;

endgroup

  function void display();
    $display("ADDR = %0d, LEN = %0d, SIZE = %0d, Memory Access = %0d",
             ADDR, LEN, SIZE, ((ADDR >> 2) + (LEN + 1)));
    $display("Delays - AW:%0d, W:%0d, B:%0d, AR:%0d, R:%0d",
             aw_valid_delay, w_valid_delay, b_ready_delay, ar_valid_delay, r_ready_delay);
  endfunction

  function new();
    axi4_cov = new();
  endfunction

endclass
```

```
`timescale 1ns/1ps
`include "axi4_class.sv"
`include "axi4_interface.sv"

module Axi4_test (axi4_interface.tb inter);

  axi4_class stim;
  logic [31:0] golden_mem [0:1023];
  logic [31:0] expected_queue[$];
  logic [31:0] actual_queue[$];
  logic [31:0] data[$];
  logic [1 :0] expected_response;
  logic [1 :0] actual_response;

  int cases = 0;
  int pass  = 0;
  int fail  = 0;
  real cov   = 0;


  bit range_mode [][2] = '{ '{1,0}, '{0,1} };   // valid, invalid range
  bit awlen_modes[][2] = '{ '{1,0}, '{0,1} };   // range, corners
  bit addr_modes [][2] = '{ '{1,0}, '{0,1} };   // range, corners
  bit data_modes [][4] = '{ '{1,0,0,0}, '{0,1,0,0}, '{0,0,1,0}, '{0,0,0,1} };

  initial begin
    stim = new();

    // Initialize golden memory
    for (int i = 0; i < 1024; i++)
      golden_mem[i] = 0;

    actual_queue.delete();
    expected_queue.delete();
    data.delete();
    reset();

```

```
    stim.delay_ranges.constraint_mode(1);
    stim.fix_size.constraint_mode(1);
    stim.aligned_address.constraint_mode(1);

    foreach (range_mode[m]) begin
    foreach (awlen_modes[i]) begin
    foreach (addr_modes[j]) begin
```

```
    cov = stim.axi4_cov.get_coverage();
    $display("=== Final Results ===");
    $display("Number of tests: %0d, Passed: %0d, Failed: %0d", cases, pass, fail);
    $display("Coverage achieved: %0.1f%%", cov);

    #200;
    $stop;
  end

  task reset();
    begin
      inter.ARESETn = 1'b1;
      @(negedge inter.ACLK);
      inter.ARESETn = 1'b0;
      @(negedge inter.ACLK);
      inter.ARESETn = 1'b1;
      @(negedge inter.ACLK);
    end
  endtask

```

```
task clear_signals();
  inter.AWVALID = 0;
  inter.WVALID  = 0;
  inter.WLAST   = 0;
  inter.BREADY  = 0;
  inter.ARVALID = 0;
  inter.RREADY  = 0;
  inter.AWADDR  = 0;
  inter.AWLEN   = 0;
  inter.AWSIZE  = 0;
  inter.WDATA   = 0;
  inter.ARADDR  = 0;
  inter.ARLEN   = 0;
  inter.ARSIZE  = 0;
  expected_response = 2'b0;
  actual_response   = 2'b0;
  @(negedge inter.ACLK);
endtask

task generate_stimulus();
  data.delete();
  stim.LEN     = 0;
  stim.ADDR    = 0;
  stim.SIZE    = 0;
  stim.DATA     = 0;
  stim.OPERATION = 0;
  stim.aw_valid_delay = 0;
  stim.w_valid_delay  = 0;
  stim.b_ready_delay  = 0;
  stim.ar_valid_delay = 0;
  stim.r_ready_delay  = 0;

  assert(stim.randomize(ADDR, LEN, SIZE, OPERATION, aw_valid_delay, w_valid_delay,
        b_ready_delay, ar_valid_delay, r_ready_delay ))
    else $fatal("Address/Length/Operation randomization failed");

```

```
  if (stim.OPERATION == 2'd1)
  begin
    for (int i = 0; i <= stim.LEN; i++)
    begin
      assert(stim.randomize(DATA))
        else $fatal("Data randomization failed for beat %0d", i);
      data[i] = stim.DATA;
      stim.axi4_cov.sample();
    end
  end
endtask

```

```
task golden_model();
  expected_queue.delete();
  if (((stim.ADDR >> stim.SIZE) + (stim.LEN + 1)) >= 1024)
  begin
      expected_response = 2'b10;
  end else
  begin
    expected_response = 2'b00;
    if (stim.OPERATION == 2'd1)
    begin
      for (int i = 0; i <= stim.LEN; i++)
```

```
task drive();
  if (stim.OPERATION == 2'd2)
  begin
    // Apply delay before starting read address phase
    repeat (stim.ar_valid_delay) @(negedge inter.ACLK);

    // Set up read address channel
    inter.ARADDR  = stim.ADDR;
    inter.ARLEN   = stim.LEN;
    inter.ARSIZE  = stim.SIZE;
    inter.ARVALID = 1'b1;

    // Wait for address acceptance
    repeat (20) @(negedge inter.ACLK)
      if (inter.ARREADY) break;
    inter.ARVALID = 0;

    collect();
  end else if (stim.OPERATION == 2'd1)
  begin
      // Apply delay before starting write address phase
      repeat (stim.aw_valid_delay) @(negedge inter.ACLK);
      // Start write address transaction

      inter.AWADDR  = stim.ADDR;
      inter.AWLEN   = stim.LEN;
      inter.AWSIZE  = stim.SIZE;
      inter.AWVALID = 1'b1;

      // Wait for address acceptance
      repeat (20) @(negedge inter.ACLK)
        if (inter.AWREADY) break;
      inter.AWVALID = 1'b0;


```

```
      if (((stim.ADDR >> stim.SIZE) + (stim.LEN + 1)) >= 1024)
      begin
          $display("TIME: $0t", $time);
          assert(stim.randomize(w_valid_delay))
            else $fatal("w_valid_delay randomization failed");
          repeat (stim.w_valid_delay) @(negedge inter.ACLK);
          stim.axi4_cov.sample();
          inter.WVALID = 1'b1;
          inter.WLAST  = 1;
          //@(negedge inter.ACLK);

```

```
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
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;
```

```
task check();
  if (stim.OPERATION == 2'd1)
  begin
    $display("Test %0d", cases);
    $display("Write Test");
    stim.display();

    if (actual_response == expected_response)
    begin
      pass++;
      $display("Test %0d PASSED", cases);
      $display("write Operation passed");
    end else
    begin
      fail++;
      $display("Test %0d FAILED Resopnse mismatch", cases);
      $display("write Operation failed");
    end

  end else
    if (stim.OPERATION == 2'd2)
    begin
      $display("Test %0d", cases);
      $display("Read Test");
      stim.display();

```

```
      if (((stim.ADDR >> stim.SIZE) + (stim.LEN + 1)) >= 1024)
      begin
          if (actual_response == expected_response)
          begin
            pass++;
            $display("Test %0d PASSED", cases);
            $display("Read Operation passed");
```

```
if ((actual_queue == expected_queue) && (actual_response == expected_response))
begin
  pass++;
  $display("Test %0d PASSED", cases);
  $display("Read Operation passed");
  $display("Expected response: %2b, Actual response: %2b",
            expected_response, actual_response);

end else
begin
  fail++;
  $display("Test %0d FAILED - Data|Resopnse mismatch", cases);
  $display("Read Operation failed");
  $display("Expected response: %2b, Actual response: %2b",
            expected_response, actual_response);

  if (actual_response != expected_response)
  begin
    $display("Expected response: %2b, Actual response: %2b",
              expected_response, actual_response);
  end

```

```
  for (int i = 0; i < expected_queue.size(); i++)
  begin
    if (expected_queue[i] != actual_queue[i])
    begin
      $display("  Beat[%0d]: Expected = %h, Actual = %h",
                i, expected_queue[i], actual_queue[i]);
    end
  end
end
```

5. Assertions

```
module axi4_assert (axi4_interface.dut inter);

  // All outputs should be properly initialized after reset
  property reset_awready;
    @(posedge inter.ACLK) !inter.ARESETn |-> inter.AWREADY == 1'b1;
  endproperty
  A_RESET_AWREADY: assert property (reset_awready)
    else $error("AWREADY not initialized to 1 after reset");
  C_RESET_AWREADY: cover property (reset_awready);

  property reset_wready;
    @(posedge inter.ACLK) !inter.ARESETn |-> inter.WREADY == 1'b0;
  endproperty
  A_RESET_WREADY: assert property (reset_wready)
    else $error("WREADY not initialized to 0 after reset");
  C_RESET_WREADY: cover property (reset_wready);

  property reset_bvalid;
    @(posedge inter.ACLK) !inter.ARESETn |-> inter.BVALID == 1'b0;
  endproperty
  A_RESET_BVALID: assert property (reset_bvalid)
    else $error("BVALID not initialized to 0 after reset");
  C_RESET_BVALID: cover property (reset_bvalid);

  property reset_arready;
    @(posedge inter.ACLK) !inter.ARESETn |-> inter.ARREADY == 1'b1;
  endproperty
  A_RESET_ARREADY: assert property (reset_arready)
    else $error("ARREADY not initialized to 1 after reset");
  C_RESET_ARREADY: cover property (reset_arready);

  property reset_rvalid;
    @(posedge inter.ACLK) !inter.ARESETn |-> inter.RVALID == 1'b0;
  endproperty
  A_RESET_RVALID: assert property (reset_rvalid)
    else $error("RVALID not initialized to 0 after reset");
  C_RESET_RVALID: cover property (reset_rvalid);
```

```
  property reset_rlast;
    @(posedge inter.ACLK) !inter.ARESETn |-> inter.RLAST == 1'b0;
  endproperty
  A_RESET_RLAST: assert property (reset_rlast)
    else $error("RLAST not initialized to 0 after reset");
  C_RESET_RLAST: cover property (reset_rlast);

  // AWREADY should go low after accepting address
  property awready_deassert;
    @(posedge inter.ACLK) disable iff (!inter.ARESETn)
    (inter.AWVALID && inter.AWREADY) |=> !inter.AWREADY;
```

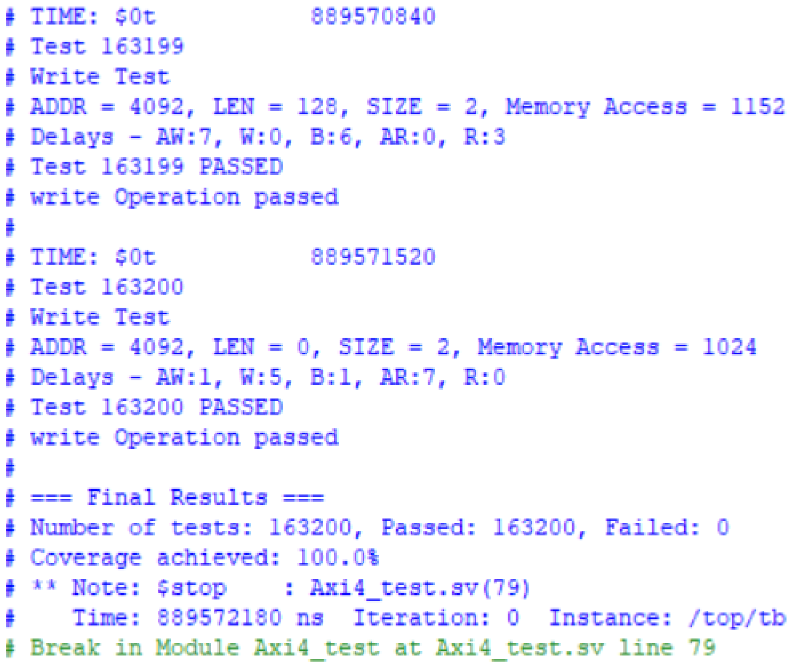
```
property awsize_stable;
  @(posedge inter.ACLK) disable iff (!inter.ARESETn)
  inter.AWVALID && !inter.AWREADY |=> $stable(inter.AWSIZE);
endproperty
A_AWSIZE_STABLE: assert property (awsize_stable)
  else $error("AWSIZE must remain stable when AWVALID is high");
C_AWSIZE_STABLE: cover property (awsize_stable);

// WDATA should remain stable when WVALID is high
property wdata_stable;
  @(posedge inter.ACLK) disable iff (!inter.ARESETn)
  inter.WVALID && !inter.WREADY |=> $stable(inter.WDATA);
endproperty
A_WDATA_STABLE: assert property (wdata_stable)
  else $error("WDATA must remain stable when WVALID is high");
C_WDATA_STABLE: cover property (wdata_stable);

// WLAST should be asserted on the last data beat
property wlast_on_last_beat;
  @(posedge inter.ACLK) disable iff (!inter.ARESETn)
  (inter.WVALID && inter.WREADY && inter.WLAST) |=> !inter.WREADY;
endproperty
A_WLAST_LAST_BEAT: assert property (wlast_on_last_beat)
  else $error("WREADY should deassert after WLAST");
C_WLAST_LAST_BEAT: cover property (wlast_on_last_beat);

// Write response must come after write data completion
property write_order;
  @(posedge inter.ACLK) disable iff (!inter.ARESETn)
  $rose(inter.BVALID) |-> $past(inter.WVALID && inter.WREADY && inter.WLAST);
endproperty
A_WRITE_ORDER_DATA_RESP: assert property (write_order)
  else $error("Write response cannot start without data completion");
// BVALID should be asserted after write data completion
property bvalid_after_wlast;
  @(posedge inter.ACLK) disable iff (!inter.ARESETn)
  (inter.WVALID && inter.WREADY && inter.WLAST) |=> inter.BVALID;
endproperty
A_BVALID_AFTER_WLAST: assert property (bvalid_after_wlast)
  else $error("BVALID should be asserted after WLAST handshake");
C_BVALID_AFTER_WLAST: cover property (bvalid_after_wlast);

// BVALID should remain stable until BREADY
property bvalid_stable;
  @(posedge inter.ACLK) disable iff (!inter.ARESETn)
  inter.BVALID && !inter.BREADY |=> inter.BVALID;
```

```
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");
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)
```

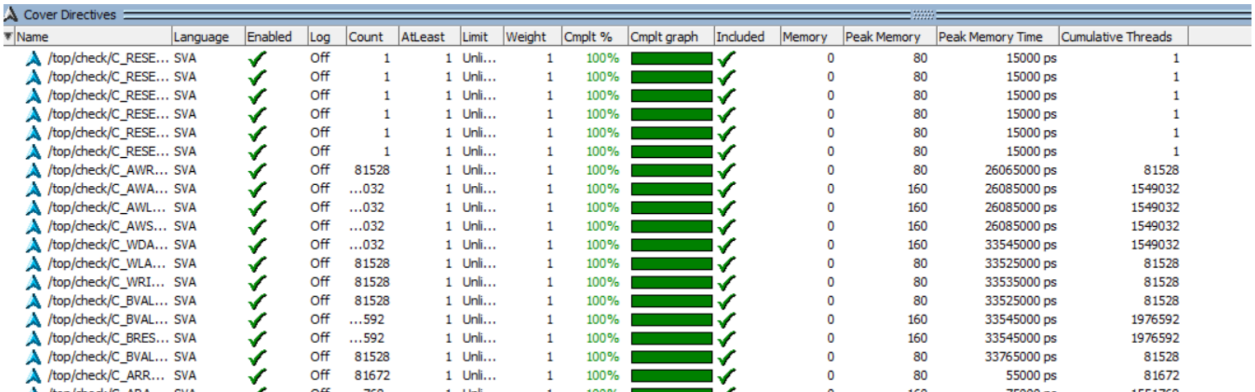
```
// BRESP should be OKAY (00) or SLVERR (10)
property bresp_valid_values;
  @(posedge inter.ACLK) disable iff (!inter.ARESETn)
  inter.BVALID |-> (inter.BRESP == 2'b00 || inter.BRESP == 2'b10);
endproperty
A_BRESP_VALID_VALUES: assert property (bresp_valid_values)
  else $error("Invalid BRESP value: %b", inter.BRESP);
C_BRESP_VALID_VALUES: cover property (bresp_valid_values);

// RRESP should be OKAY (00) or SLVERR (10)
property rresp_valid_values;
  @(posedge inter.ACLK) disable iff (!inter.ARESETn)
  inter.RVALID |-> (inter.RRESP == 2'b00 || inter.RRESP == 2'b10);
endproperty
A_RRESP_VALID_VALUES: assert property (rresp_valid_values)
  else $error("Invalid RRESP value: %b", inter.RRESP);
C_RRESP_VALID_VALUES: cover property (rresp_valid_values);

// 4KB boundary crossing should result in SLVERR for write
property write_boundary;
  @(posedge inter.ACLK) disable iff (!inter.ARESETn)
  (inter.BVALID &&
   (((inter.AWADDR & 16'h0FFF) + ((inter.AWLEN + 1) << inter.AWSIZE)) > 16'h0FFF))
  |-> inter.BRESP == 2'b10;
endproperty
A_WRITE_BOUNDARY_ERROR: assert property (write_boundary)
  else $error("4KB boundary crossing should result in SLVERR");
C_WRITE_BOUNDARY_ERROR: cover property (write_boundary);
```

```
// Out of memory range should result in SLVERR for write
property write_range;
  @(posedge inter.ACLK) disable iff (!inter.ARESETn)
  (inter.BVALID && ((inter.AWADDR >> 2) >= 1024))
  |-> inter.BRESP == 2'b10;
endproperty
A_WRITE_RANGE_ERROR: assert property (write_range)
  else $error("Out of range write should result in SLVERR");
C_WRITE_RANGE_ERROR: cover property (write_range);

// 4KB boundary crossing should result in SLVERR for read
property read_boundary;
  @(posedge inter.ACLK) disable iff (!inter.ARESETn)
  (inter.RVALID &&
```

IV. System Results

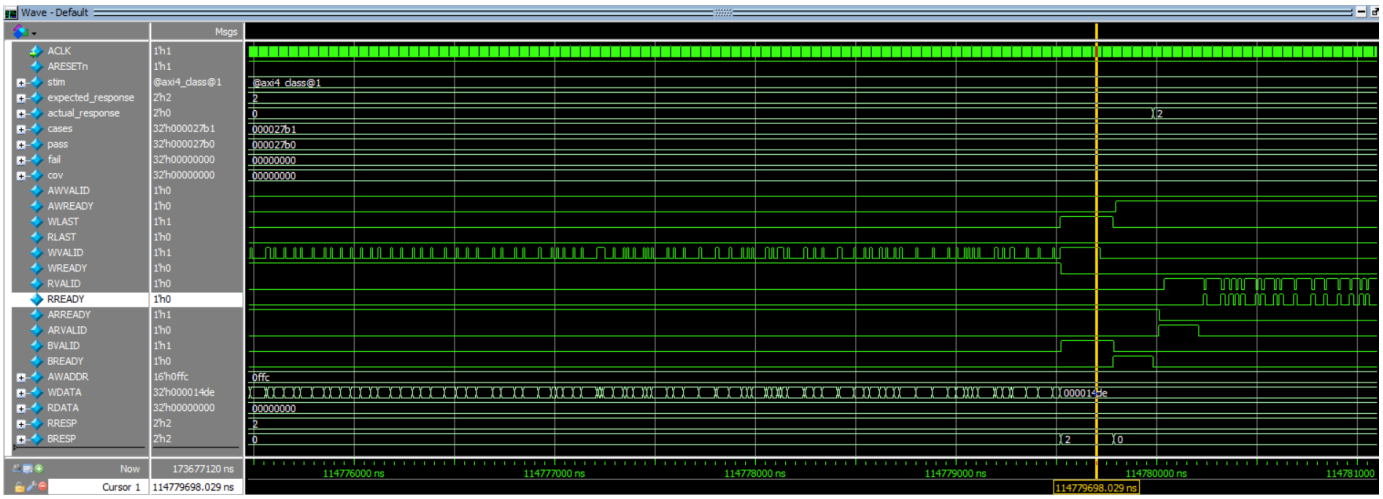


```
# TIME: $0t           889570840
# Test 163199
# Write Test
# ADDR = 4092, LEN = 128, SIZE = 2, Memory Access = 1152
# Delays - AW:7, W:0, B:6, AR:0, R:3
# Test 163199 PASSED
# write Operation passed
#
# TIME: $0t           889571520
# Test 163200
# Write Test
# ADDR = 4092, LEN = 0, SIZE = 2, Memory 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
```

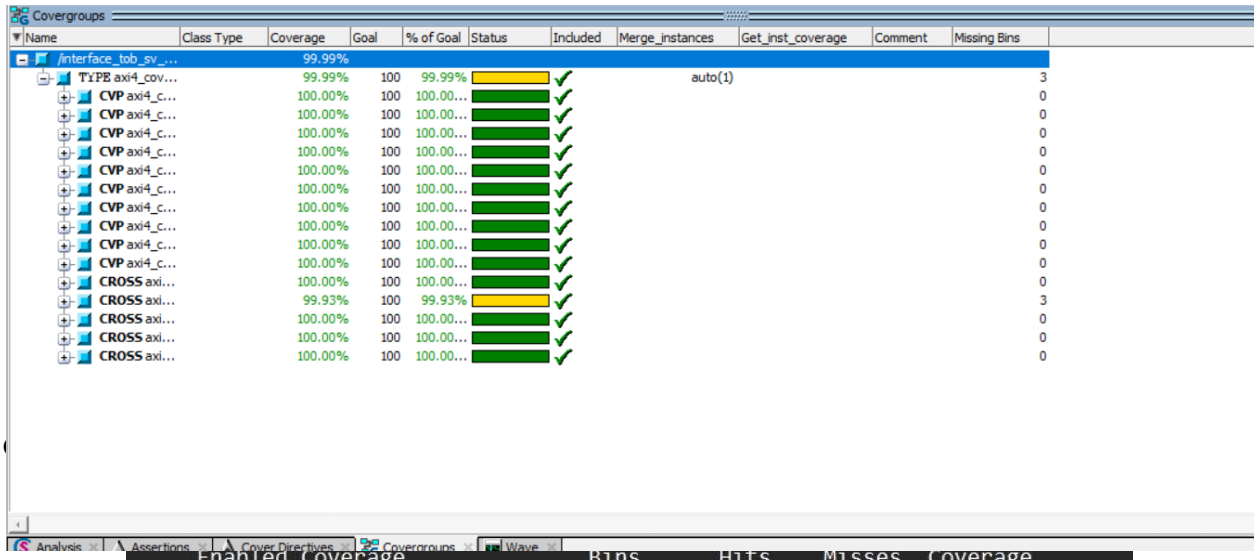


Cover Directives

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



Function coverage



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              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       -
```

- **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

```
vlib work
vlog *.sv
vsim -assertdebug +acc -voptargs=+acc work.top
do wave.do

run -all

```

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

```