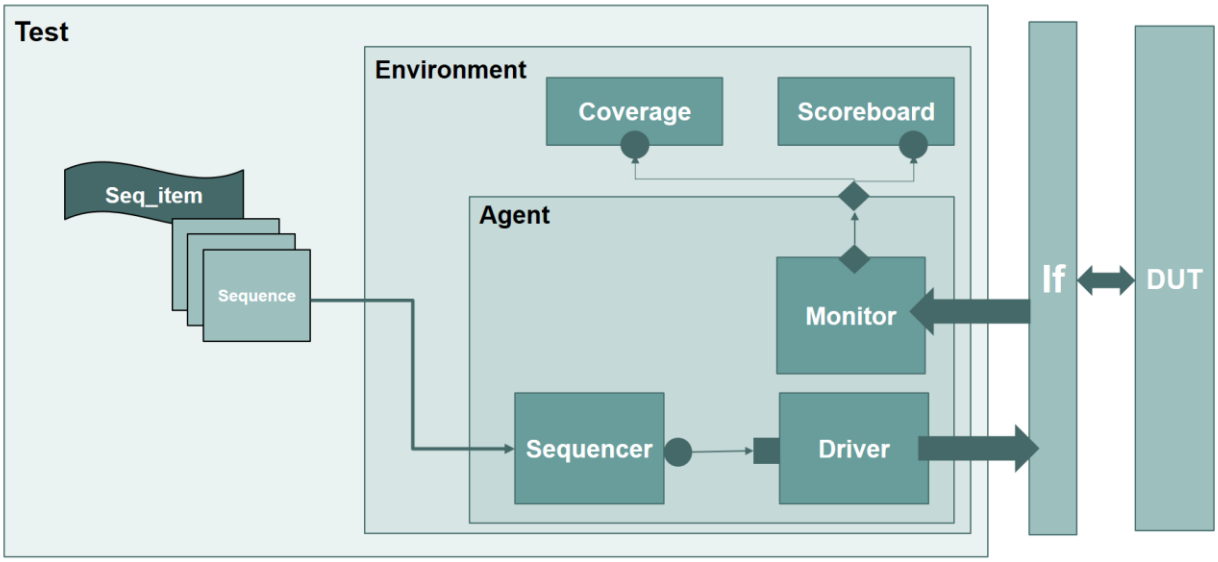
AXI4 Verification Project

Abdelrahman Ahmed

Contents

Project Overview:

This report presents the comprehensive verification of an AXI4-compliant memory-mapped slave design using the Universal Verification Methodology (UVM). The project demonstrates the practical application of advanced verification concepts and methodologies to ensure the reliability and functionality of a critical digital design component.



AXI4 System:

The Advanced eXtensible Interface (AXI4) protocol, developed by ARM, has become the industry standard for high-performance, high-frequency system designs. Memory-mapped slave devices implementing this protocol are fundamental components in modern System-on-Chip (SoC) architectures, requiring rigorous verification to ensure correct functionality across various operating conditions and use cases.

The complexity of AXI4 protocol compliance, combined with the critical nature of memory operations in digital systems, necessitates a systematic and comprehensive verification approach. Traditional verification methodologies often fall short in providing the coverage depth and reusability required for such complex designs. This project addresses these challenges by implementing a complete UVM-based verification environment.

Objectives:

The primary objectives of this verification project are:

1. **Complete UVM Environment Development**: Implement a full-scale UVM verification environment incorporating all essential components including agents, drivers, monitors, sequencers, sequences, and scoreboards.
2. **Comprehensive Functional Verification**: Verify the core READ and WRITE operations of the AXI4 memory-mapped slave design, ensuring protocol compliance and correct data handling.
3. **Coverage Closure**: Achieve 100% functional and code coverage with appropriate justifications for any coverage gaps, demonstrating thorough verification completeness.
4. **Assertion-Based Verification**: Integrate SystemVerilog Assertions (SVA) within the UVM framework to provide real-time protocol checking and error detection.
5. **UVM Best Practices Implementation**: Demonstrate proper usage of UVM phasing, reporting mechanisms, and testbench architecture following industry best practices.

Scope and Approach:

This project focuses on verifying the fundamental memory operations (READ/WRITE) of the AXI4-compliant design while maintaining the flexibility to extend verification to burst operations and advanced features. The verification approach employs:

- **Layered Testbench Architecture**: Implementation of a scalable UVM testbench structure with clear separation of concerns
- **Constrained Random Verification**: Generation of comprehensive stimulus patterns using SystemVerilog constraints
- **Coverage-Driven Verification**: Systematic tracking of functional and code coverage metrics to ensure verification completeness
- **Assertion-Based Verification**: Integration of protocol-specific assertions for real-time checking
- **Advanced UVM Features**: Utilization of factory patterns, configuration objects, and TLM communicatio

Design Under Test:

The Design Under Test (DUT) is an AXI4-compliant memory-mapped slave that interfaces with AXI4 masters to handle memory transactions. The design implements the standard AXI4 protocol features including:

- Address and data channel handshaking
- Read and write transaction processing
- Response generation and error handling
- Protocol-compliant signal timing and behavior

- AXI4 Memory:

```
module axi4_memory #(
    parameter DATA_WIDTH = 32,
    parameter ADDR_WIDTH = 10,    // For 1024 locations
    parameter DEPTH = 1024
)(
    input  wire                   clk,
    input  wire                   rst_n,

    input  wire                   mem_en,
    input  wire                   mem_we,
    input  wire [ADDR_WIDTH-1:0]  mem_addr,
    input  wire [DATA_WIDTH-1:0]  mem_wdata,
    output reg  [DATA_WIDTH-1:0]  mem_rdata
);

    // Memory array
    reg [DATA_WIDTH-1:0] memory [0:DEPTH-1];


    integer j;

    // 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
            begin
                mem_rdata <= memory[mem_addr];
            end
        end
    end

    // Initialize memory
    initial begin
        for (j = 0; j < DEPTH; j = j + 1)
            memory[j] = 0;
    end

endmodule
```

AXI4 design:

```
module axi4 #(
    parameter DATA_WIDTH = 32,
    parameter ADDR_WIDTH = 16,
    parameter MEMORY_DEPTH = 1024
)(
    input  wire                   ACLK,
    input  wire                   ARESETn,

    // Write address channel
    input  wire [ADDR_WIDTH-1:0]  AWADDR,
    input  wire [7:0]             AWLEN,
    input  wire [2:0]             AWSIZE,
    input  wire                   AWVALID,
    output reg                    AWREADY,

    // Write data channel
    input  wire [DATA_WIDTH-1:0]  WDATA,
    input  wire                   WVALID,
    input  wire                   WLAST,
    output reg                    WREADY,

    // Write response channel
    output reg [1:0]              BRESP,
    output reg                    BVALID,
    input  wire                   BREADY,

    // Read address channel
    input  wire [ADDR_WIDTH-1:0]  ARADDR,
    input  wire [7:0]             ARLEN,
    input  wire [2:0]             ARSIZE,
    input  wire                   ARVALID,
    output reg                    ARREADY,

    // Read data channel
    output reg [DATA_WIDTH-1:0]   RDATA,
    output reg [1:0]              RRESP,
    output reg                    RVALID,
    output reg                    RLAST,
    input  wire                   RREADY
);


    // Internal memory signals
    reg mem_en, mem_we;
    reg [$clog2(MEMORY_DEPTH)-1:0] mem_addr;
    reg [DATA_WIDTH-1:0] mem_wdata;
    wire [DATA_WIDTH-1:0] mem_rdata;

    // Address and burst management
    reg [ADDR_WIDTH-1:0] write_addr, read_addr;
    reg [7:0] write_burst_len, read_burst_len;
    reg [7:0] write_burst_cnt, read_burst_cnt;
    reg [2:0] write_size, read_size;
```

```
    wire [ADDR_WIDTH-1:0] write_addr_incr,read_addr_incr;


    wire write_boundary_cross, read_boundary_cross;
    wire write_addr_valid, read_addr_valid;



    // Address increment calculation
    assign  write_addr_incr = (1 << write_size);
    assign  read_addr_incr  = (1 << read_size);

    // Address boundary check (4KB boundary = 12 bits)
    assign write_boundary_cross = ((AWADDR & 12'hFFF) + ((AWLEN) << AWSIZE)) > 12'hFFF;
    assign read_boundary_cross  = ((ARADDR & 12'hFFF) + ((ARLEN) << ARSIZE)) > 12'hFFF;

    // Address range check
    assign write_addr_valid = (write_addr >> 2) < MEMORY_DEPTH;
    assign read_addr_valid = (read_addr >> 2) < MEMORY_DEPTH;

    // Memory instance
    axi4_memory #(
        .DATA_WIDTH(DATA_WIDTH),
        .ADDR_WIDTH($clog2(MEMORY_DEPTH)),
        .DEPTH(MEMORY_DEPTH)
    ) mem_inst (
        .clk(ACLK),
        .rst_n(ARESETn),
        .mem_en(mem_en),
        .mem_we(mem_we),
        .mem_addr(mem_addr),
        .mem_wdata(mem_wdata),
        .mem_rdata(mem_rdata)
    );

    // FSM states
    reg [2:0] write_state;
    localparam W_IDLE = 3'd0,
               W_ADDR = 3'd1,
               W_DATA = 3'd2,
               W_RESP = 3'd3;

    reg [2:0] read_state;
    localparam R_IDLE = 3'd0,
               R_ADDR = 3'd1,
               R_DATA = 3'd2;

    // Registered memory read data for timing
    reg [DATA_WIDTH-1:0] mem_rdata_reg;
```

```
        always @(posedge ACLK or negedge ARESETn) begin
            if (!ARESETn) begin
                // Reset all outputs
                AWREADY <= 1'b1;  // Ready to accept address
                WREADY <= 1'b0;
                BVALID <= 1'b0;
                BRESP <= 2'b00;

                ARREADY <= 1'b1;  // Ready to accept address
                RVALID <= 1'b0;
                RRESP <= 2'b00;
                RDATA <= {DATA_WIDTH{1'b0}};
                RLAST <= 1'b0;

                // Internal reset
                write_state <= W_IDLE;
                read_state  <= R_IDLE;

                mem_en      <= 1'b0;
                mem_we      <= 1'b0;
                mem_addr    <= {$clog2(MEMORY_DEPTH){1'b0}};
                mem_wdata   <= {DATA_WIDTH{1'b0}};

                write_addr        <= {ADDR_WIDTH{1'b0}};
                write_burst_len   <= 8'b0;
                write_burst_cnt   <= 8'b0;
                write_size        <= 3'b0;

                read_addr         <= {ADDR_WIDTH{1'b0}};
                read_burst_len    <= 8'b0;
                read_burst_cnt    <= 8'b0;
                read_size         <= 3'b0;

                mem_rdata_reg <= {DATA_WIDTH{1'b0}};

            end else begin
                // Default memory disable
                mem_en <= 1'b0;
                mem_we <= 1'b0;
```

```
            // --------------------------
            // Write Channel FSM
            // --------------------------
            case (write_state)
                W_IDLE: begin
                    AWREADY <= 1'b1;
                    WREADY <= 1'b0;
                    BVALID <= 1'b0;

                    if (AWVALID && AWREADY) begin
                        // Capture address phase information
                        write_addr <= AWADDR;
                        write_burst_len <= AWLEN;
                        write_burst_cnt <= AWLEN;
                        write_size <= AWSIZE;

                        AWREADY <= 1'b0;
                        write_state <= W_ADDR;
                    end
                end

                W_ADDR: begin
                    // Transition to data phase
                    WREADY <= 1'b1;
                    write_state <= W_DATA;
                end

                W_DATA: begin
                    if (WVALID && WREADY) begin
                        // Check if address is valid
                        if (write_addr_valid && !write_boundary_cross)
                        begin
                            // Perform write operation
                            mem_en <= 1'b1;
                            mem_we <= 1'b1;
                            mem_addr <= write_addr >> 2;  // Convert to word address
                            mem_wdata <= WDATA;
                        end

                        // Check for last transfer
                        if (WLAST || write_burst_cnt == 0)
                        begin
                            WREADY <= 1'b0;
                            write_state <= W_RESP;

                            // Set response - delayed until write completion
                            if (!write_addr_valid || write_boundary_cross) begin
                                BRESP <= 2'b10;  // SLVERR
                            end else begin
                                BRESP <= 2'b00;  // OKAY
                            end
                            BVALID <= 1'b1;
```

```
                        // Set response - delayed until write completion
                        if (!write_addr_valid || write_boundary_cross) begin
                            BRESP <= 2'b10;  // SLVERR
                        end else begin
                            BRESP <= 2'b00;  // OKAY
                        end
                        BVALID <= 1'b1;
                    end else
                    begin
                        // Continue burst - increment address
                        write_addr <= write_addr + write_addr_incr;
                        write_burst_cnt <= write_burst_cnt - 1'b1;
                    end
                end
            end

            W_RESP: begin
                if (BREADY && BVALID) begin
                    BVALID <= 1'b0;
                    BRESP <= 2'b00;
                    write_state <= W_IDLE;
                end
            end

            default: write_state <= W_IDLE;
        endcase

```

```
216
217                 // --------------------------
218                 // Read Channel FSM
219                 // --------------------------
220                 case (read_state)
221                     R_IDLE: begin
222                         ARREADY <= 1'b1;
223                         RVALID <= 1'b0;
224                         RLAST <= 1'b0;
225
226                         if (ARVALID && ARREADY)
227                         begin
228                             // Capture address phase information
229                             read_addr <= ARADDR;
230                             read_burst_len <= ARLEN;
231                             read_burst_cnt <= ARLEN;
232                             read_size <= ARSIZE;
233
234                             ARREADY <= 1'b0;
235                             read_state <= R_ADDR;
236                         end
237                     end
238
239                     R_ADDR: begin
240                         // Start first read
241                         if (read_addr_valid && !read_boundary_cross)
242                         begin
243                             mem_en <= 1'b1;
244                             mem_addr <= read_addr >> 2;  // Convert to word address
245                         end
246                         read_state <= R_DATA;
247                     end
248
249                     R_DATA: begin
250                         // Present read data
251                         if (read_addr_valid && !read_boundary_cross) begin
252                             RDATA <= mem_rdata;
253                             RRESP <= 2'b00;  // OKAY
254                         end else begin
255                             RDATA <= {DATA_WIDTH{1'b0}};
256                             RRESP <= 2'b10;  // SLVERR
257                         end
258
259                         RVALID <= 1'b1;
260                         RLAST <= (read_burst_cnt == 0);
261
262                         if (RREADY && RVALID)
263                         begin
264                             RVALID <= 1'b0;
265
```

```
                            if (read_burst_cnt > 0)
                            begin
                                // Continue burst
                                read_addr <= read_addr + read_addr_incr;
                                read_burst_cnt <= read_burst_cnt - 1'b1;

                                // Start next read
                                if (read_addr_valid && !read_boundary_cross) begin
                                    mem_en <= 1'b1;
                                    mem_addr <= (read_addr + read_addr_incr) >> 2;
                                end

                                // Stay in R_DATA for next transfer
                            end else
                            begin
                                // End of burst
                                RLAST <= 1'b0;
                                read_state <= R_IDLE;
                            end
                        end
                    end

                    default: read_state <= R_IDLE;
                endcase
            end
        end

    endmodule
```

Interface:

```
interface axi4_if ();

    logic        ACLK;
    logic        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 [15:0] ARADDR;
    logic [31:0] RDATA;
    logic [7:0]  ARLEN;
    logic [2:0]  ARSIZE;
    logic [1:0]  RRESP;
    logic [1:0]  BRESP;

endinterface
```

Design top:

```
`timescale 1ns/1ps
`include "axi4_if.sv"
`include "axi4_pkg.sv"
`include "axi4_assert.sv"
`include "uvm_macros.svh"

import uvm_pkg::*;
import axi4_pkg::*;

module top ();

    axi4_if axi4_vif();
    axi4_assert check (axi4_vif);

    // DUT instantiation with proper port connections
    axi4 #(
        .DATA_WIDTH(32),
        .ADDR_WIDTH(16),
        .MEMORY_DEPTH(1024)
    ) DUT (
        .ACLK    (axi4_vif.ACLK),
        .ARESETn(axi4_vif.ARESETn),
        .AWADDR (axi4_vif.AWADDR),
        .AWLEN   (axi4_vif.AWLEN),
        .AWSIZE  (axi4_vif.AWSIZE),
        .AWVALID(axi4_vif.AWVALID),
        .AWREADY(axi4_vif.AWREADY),
        .WDATA   (axi4_vif.WDATA),
        .WVALID  (axi4_vif.WVALID),
        .WLAST   (axi4_vif.WLAST),
        .WREADY  (axi4_vif.WREADY),
        .BRESP   (axi4_vif.BRESP),
        .BVALID  (axi4_vif.BVALID),
        .BREADY  (axi4_vif.BREADY),
        .ARADDR  (axi4_vif.ARADDR),
        .ARLEN   (axi4_vif.ARLEN),
        .ARSIZE  (axi4_vif.ARSIZE),
        .ARVALID(axi4_vif.ARVALID),
        .ARREADY(axi4_vif.ARREADY),
        .RDATA   (axi4_vif.RDATA),
        .RRESP   (axi4_vif.RRESP),
        .RVALID  (axi4_vif.RVALID),
        .RLAST   (axi4_vif.RLAST),
        .RREADY  (axi4_vif.RREADY)
    );

    initial
    begin
        axi4_vif.ACLK = 0;
        forever
        begin
            #5ns axi4_vif.ACLK = ~axi4_vif.ACLK;
        end
    end
```

```
    initial
    begin
        uvm_config_db#(uvm_active_passive_enum)::set(null, "uvm_test_top.env.agt","is_active", UVM_ACTIVE);
        /*uvm_config_db#(virtual axi4_if)::set(null, "uvm_test_top.env.agt.*", "axi4_intf", axi4_vif);
        uvm_config_db#(virtual axi4_if)::set(null, "uvm_test_top.env.scb", "axi4_intf", axi4_vif);
        uvm_config_db#(virtual axi4_if)::set(null, "uvm_test_top.env.agt.sqr.*", "axi4_intf", axi4_vif);*/
        uvm_config_db#(virtual axi4_if)::set(null, "*", "axi4_intf", axi4_vif);

        run_test("axi4_test");
    end

endmodule
```

Sequnce item:

- Declerations:

```
`ifndef AXI4_TRANSACTION_SVH
`define AXI4_TRANSACTION_SVH

`include "uvm_macros.svh"
import uvm_pkg::*;

class axi4_transaction extends uvm_sequence_item;

  // 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;


    logic [31:0] all_data[$];
    logic [31:0] actual_queue[$];
    logic [1 :0] actual_response;


    // Random delays for handshaking
    rand  logic [2:0] aw_valid_delay;
    rand  logic [2:0] w_valid_delay;
    rand  logic [2:0] b_ready_delay;
    rand  logic [2:0] ar_valid_delay;
    rand  logic [2:0] r_ready_delay;

    rand bit below_1024;
    rand int unsigned data_mode;
    rand int unsigned addr_mode;
    rand int unsigned len_mode;

```

- constraints:

```
// Constraints
constraint OPERATION_C {
  OPERATION dist { 2'd1 := 50, 2'd2 := 50 }; // Equal distribution
}

// Delay constraints
constraint delay_ranges {
  aw_valid_delay inside {[0:7]};
  w_valid_delay  inside {[0:7]};
  b_ready_delay  inside {[0:7]};
  ar_valid_delay inside {[0:7]};
  r_ready_delay  inside {[0:7]};
}

constraint fix_size {
  SIZE == 2;  // Fixed to 2 (4 bytes)
}


constraint aligned_address {
  ADDR[1:0] == 2'b00; // Aligned to 4-byte boundary
}


// Valid access constraint
constraint range_split {
below_1024 dist {1 := 50, 0 := 50};
if (below_1024)
  ((ADDR >> 2) + (LEN + 1)) <= 1024;
else
  ((ADDR >> 2) + (LEN + 1)) > 1024;
}


constraint len_mode_dist {
len_mode dist {0 := 80, 1 := 20};
}


constraint LEN_range_c {
  if (len_mode == 0) {
    LEN inside {[8'd0 : 8'd255]};
  }
}

```

```
  constraint LEN_corners_c {
    if (len_mode == 1) {
      LEN inside {
        8'd0,
        8'd1,
        8'd127,
        8'd128,
        8'd254,
        8'd255
      };
    }
  }



  constraint addr_mode_dist {
  addr_mode dist {0 := 90, 1 := 10};
  }

  constraint addr_c {
  if (addr_mode == 0) {
    ADDR inside {[16'd0 : 16'd65535]};
    }
  }

  constraint addr_corner_c {
    if (addr_mode == 1) {
      ADDR inside {
        16'd0,
        16'd1024,
        16'd2048,
        16'd4092,
        16'b1111_1111_1100
      };
    }
  }



  constraint data_mode_dist {
  data_mode dist {0:=30, 1:=30, 2:=30, 3:=10};
  }

  constraint data_c1 {
  if (data_mode == 0) {
    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 {
    if (data_mode == 1) {
      DATA inside {
        [32'd0          : 32'd1],
        [32'd32768      : 32'd49151],
        [32'd49152      : 32'd65535],
        [32'd65536      : 32'd262143],
        [32'd262144     : 32'd1048575]
      };
    }
  }

  constraint data_c3 {
    if (data_mode == 2) {
      DATA inside {
        [32'd0          : 32'd1],
        [32'd1048576    : 32'd16777215],
        [32'd16777216   : 32'd268435455],
        [32'd268435456  : 32'd1073741823],
        [32'd1073741824 : 32'hFFFFFFFF]
      };
    }
  }

  constraint data_corner_c {
    if (data_mode == 3) {
      DATA inside {
        32'd0,
        32'd1,
        32'hFFFF_FFFF,
        32'hAAAA_AAAA,
        32'h1111_0000,
        32'h0000_1111
      };
    }
  }
```

- Functions:

```
    function void display();
      `uvm_info(get_type_name(), $sformatf("ADDR = %0d, LEN = %0d, SIZE = %0d, Memory Access = %0d",
                        ADDR, LEN, SIZE, ((ADDR >> 2) + (LEN))), UVM_LOW)

      `uvm_info(get_type_name(), $sformatf("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), UVM_LOW)
    endfunction


    `uvm_object_utils_begin(axi4_transaction)
        `uvm_field_int(ADDR,       UVM_DEFAULT)
        `uvm_field_int(LEN,        UVM_DEFAULT)
        `uvm_field_int(SIZE,       UVM_DEFAULT)
        `uvm_field_int(DATA,       UVM_DEFAULT)
        `uvm_field_int(OPERATION,  UVM_DEFAULT)

        // Handshake delays
        `uvm_field_int(aw_valid_delay, UVM_DEFAULT)
        `uvm_field_int(w_valid_delay,  UVM_DEFAULT)
        `uvm_field_int(b_ready_delay,  UVM_DEFAULT)
        `uvm_field_int(ar_valid_delay, UVM_DEFAULT)
        `uvm_field_int(r_ready_delay,  UVM_DEFAULT)
    `uvm_object_utils_end


    function new(string name = "axi4_transaction");
      super.new(name);
      //axi4_cov = new(this);  // Construct the covergroup
      `uvm_info("axi4_transaction", "INSIDE NEW TRANSACTION CLASS", UVM_LOW)
    endfunction

  endclass

  `endif
```

Main Sequence:

```
class axi4_sequence extends uvm_sequence #(axi4_transaction);

    `uvm_object_utils(axi4_sequence)

    int transaction_count = 0;

    common_cfg m_cfg;

    function new(string name = "axi4_sequence");
    begin

        super.new(name);
        `uvm_info("axi4_sequence", "INSIDE NEW SEQUENCE CLASS", UVM_LOW)
    end
    endfunction


    task body();

    axi4_transaction req;
    axi4_sequencer sqr;

        $cast(sqr, m_sequencer);
        m_cfg = sqr.m_cfg;

    repeat(500000)
    begin
        transaction_count++;  // Increment counter
        req = axi4_transaction::type_id::create("req");
        req.all_data.delete();

        start_item(req);
        assert(req.randomize())
           else `uvm_fatal(get_type_name(), "Transaction randomization failed")

        // Debug: Print what operation was generated
        /*`uvm_info(get_type_name(),$sformatf("Transaction %0d: OPERATION = %0d (%s)",
        transaction_count,
        req.OPERATION,
        (req.OPERATION == 2'd1) ? "WRITE" : "READ"),
        UVM_LOW)*/

        finish_item(req);

        @ (m_cfg.monitor_sent_e);

        //`uvm_info(get_type_name(), $sformatf("Transaction %0d completed", transaction_count), UVM_LOW)
    end

    `uvm_info(get_type_name(), $sformatf("Sequence completed - Generated %0d transactions", transaction_count), UVM_LOW)

    endtask
```

Debug sequence:

```
class debug_sequence extends uvm_sequence #(axi4_transaction);
    `uvm_object_utils(debug_sequence)

    int transaction_count = 0;

    common_cfg m_cfg;

    // Known failing scenarios based on your log pattern
    typedef struct {
        bit [15:0] addr;
        bit [7:0]  len;
        bit [2:0]  size;
        bit [1:0]  operation;
        string     description;
    } debug_scenario_t;

    debug_scenario_t failing_scenarios[] = '{
        // Scenarios that commonly cause failures
        '{addr: 16'h0FF0, len: 8'd3,   size: 3'd2, operation: 2'd2, description: "Read near 4KB boundary"},
        '{addr: 16'h0FFC, len: 8'd1,   size: 3'd2, operation: 2'd2, description: "Read at 4KB boundary"},
        '{addr: 16'h1000, len: 8'd255, size: 3'd2, operation: 2'd1, description: "Write with max length crossing boundary"},
        '{addr: 16'h0000, len: 8'd0,   size: 3'd2, operation: 2'd2, description: "Minimum read length"},
        '{addr: 16'hFFFC, len: 8'd255, size: 3'd2, operation: 2'd2, description: "Max address with long burst"},
        '{addr: 16'h0FE0, len: 8'd7,   size: 3'd2, operation: 2'd1, description: "Write crossing boundary"},
        '{addr: 16'h0FE4, len: 8'd6,   size: 3'd2, operation: 2'd2, description: "Read crossing boundary"},
        '{addr: 16'h0FF8, len: 8'd2,   size: 3'd2, operation: 2'd2, description: "Read exactly at boundary"},
        '{addr: 16'h1004, len: 8'd1,   size: 3'd2, operation: 2'd1, description: "Write just after boundary"},
        '{addr: 16'h0800, len: 8'd127, size: 3'd2, operation: 2'd2, description: "Large read in middle range"}
    };

    function new(string name = "debug_sequence");
        super.new(name);
        `uvm_info("debug_sequence", "INSIDE NEW DEBUG SEQUENCE CLASS", UVM_LOW)
    endfunction

    task body();
        axi4_transaction req;
        axi4_sequencer sqr;

        $cast(sqr, m_sequencer);
        m_cfg = sqr.m_cfg;

        `uvm_info(get_type_name(), "Starting debug sequence with targeted scenarios", UVM_LOW)

        // Run each specific scenario multiple times
        foreach(failing_scenarios[i]) begin
            `uvm_info(get_type_name(),
                $sformatf("Testing scenario %0d: %s", i, failing_scenarios[i].description), UVM_LOW)
```

```
            repeat(50) begin  // Run each scenario 50 times
                transaction_count++;
                req = axi4_transaction::type_id::create("req");
                req.all_data.delete();

                start_item(req);

                // Disable randomization for main signals
                req.ADDR.rand_mode(0);
                req.LEN.rand_mode(0);
                req.SIZE.rand_mode(0);
                req.OPERATION.rand_mode(0);

                // Disable conflicting constraints
                req.constraint_mode(0);  // disable ALL constraints

                // Apply fixed values from scenario
                req.ADDR      = failing_scenarios[i].addr;
                req.LEN       = failing_scenarios[i].len;
                req.SIZE      = failing_scenarios[i].size;
                req.OPERATION = failing_scenarios[i].operation;

                // Randomize only handshake delays
                assert(req.randomize(aw_valid_delay, w_valid_delay, b_ready_delay,
                                     ar_valid_delay, r_ready_delay))
                  else `uvm_fatal(get_type_name(), "Delay randomization failed")

                // Generate data for write operations
                if (req.OPERATION == 2'd1) begin
                    for (int j = 0; j <= req.LEN; j++) begin
                        assert(req.randomize(DATA))
                            else `uvm_fatal(get_type_name(),
                                $sformatf("Data randomization failed for beat %0d", j))
                        req.all_data.push_back(req.DATA);
                        //req.axi4_cov.sample();
                    end
                end else begin
                    //req.axi4_cov.sample();
                end

                finish_item(req);
                @ (m_cfg.monitor_sent_e);
            end

            `uvm_info(get_type_name(),
                $sformatf("Completed scenario %0d: %s", i, failing_scenarios[i].description), UVM_LOW)
        end


        `uvm_info(get_type_name(),
            $sformatf("Debug sequence completed - Generated %0d targeted transactions",
            transaction_count), UVM_LOW)
    endtask
```

AXI4_Test:

```
class axi4_test extends uvm_test;

    axi4_env       env;
    axi4_sequence  seq;
    debug_sequence debug;
    common_cfg     m_cfg;

    `uvm_component_utils(axi4_test)

    function new(string name = "axi4_test", uvm_component parent = null);
    begin
        super.new(name,parent);
        `uvm_info("axi4_test", "INSIDE NEW TEST CLASS", UVM_LOW)
    end
    endfunction

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        env   = axi4_env::type_id::create("env", this);
        seq   = axi4_sequence::type_id::create("seq", this);
        debug = debug_sequence::type_id::create("debug", this);
        m_cfg = common_cfg::type_id::create("m_cfg");
        `uvm_info(get_type_name(), "axi4 test build phase", UVM_LOW)

        uvm_config_db#(common_cfg)::set(this, "*", "m_cfg", m_cfg);
    endfunction


    function void end_of_elaboration_phase(uvm_phase phase);
        super.end_of_elaboration_phase(phase);
        uvm_top.print_topology();
    endfunction

    task run_phase(uvm_phase phase);
        phase.raise_objection(this);
        `uvm_info(get_type_name(), "Starting AXI4 test sequence", UVM_LOW)
        fork
            begin
                seq.start(env.agt.sqr);
                `uvm_info(get_type_name(), "AXI4 test sequence completed", UVM_LOW)

                debug.start(env.agt.sqr);
                `uvm_info(get_type_name(), "Debug sequence completed", UVM_LOW)
            end
        join

        phase.drop_objection(this);
    endtask


endclass

`endif
```

AXI4_Driver:

```
class axi4_driver extends uvm_driver #(axi4_transaction) ;

    virtual axi4_if axi4_vif;
    common_cfg m_cfg;

    uvm_analysis_port #(axi4_transaction) ap;

    `uvm_component_utils(axi4_driver)

    static int driver_transaction_count = 0;

    function new (string name = "axi4_driver", uvm_component parent = null);
        super.new(name,parent);
        ap = new("ap", this); // instantiate analysis port
        `uvm_info("axi4_driver", "INSIDE NEW DRIVER CLASS", UVM_LOW)
    endfunction

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        `uvm_info(get_type_name(), "axi4 driver build phase", UVM_LOW)

        if(!uvm_config_db#(virtual axi4_if)::get(this, "*", "axi4_intf", axi4_vif ))
            `uvm_fatal(get_full_name(), {"virtual axi4_vifface must be set for:", ".axi4_vif"})
        else
            $display("configuration DB dond");

        if(!uvm_config_db#(common_cfg)::get(this, "*", "m_cfg", m_cfg))
            `uvm_fatal(get_full_name(), "Failed to get common_cfg from config DB")
        else
            $display("common_cfg retrieved successfully in driver");
    endfunction


    task run_phase(uvm_phase phase);

        reset_signals();

        forever
        begin

            axi4_transaction req;

            // In driver's run_phase, add a counter
            /*driver_transaction_count++;
            `uvm_info(get_type_name(), $sformatf("Driver processing transaction %0d", driver_transaction_count), UVM_LOW)*/

            seq_item_port.get_next_item(req);
            ap.write(req);
            drive(req);
            seq_item_port.item_done();
        end
    endtask

```

```
    task reset_signals();
        axi4_vif.ARESETn = 1'b0;
        axi4_vif.AWVALID = 0;
        axi4_vif.WVALID  = 0;
        axi4_vif.WLAST   = 0;
        axi4_vif.BREADY  = 0;
        axi4_vif.ARVALID = 0;
        axi4_vif.RREADY  = 0;
        axi4_vif.AWADDR  = 0;
        axi4_vif.AWLEN   = 0;
        axi4_vif.AWSIZE  = 0;
        axi4_vif.WDATA   = 0;
        axi4_vif.ARADDR  = 0;
        axi4_vif.ARLEN   = 0;
        axi4_vif.ARSIZE  = 0;

        // Hold reset for multiple clock cycles
        repeat(3) @(posedge axi4_vif.ACLK);
        axi4_vif.ARESETn = 1'b1;
        // Wait for reset deassertion to settle
        repeat(2) @(posedge axi4_vif.ACLK);
    endtask

    extern task drive(axi4_transaction req);

endclass


task axi4_driver::drive(axi4_transaction req);

    if (req.OPERATION == 2'd2)
    begin
      // Apply delay before starting read address phase
      repeat (req.ar_valid_delay) @(negedge axi4_vif.ACLK);

      // Set up read address channel
      axi4_vif.ARADDR  = req.ADDR;
      axi4_vif.ARLEN   = req.LEN;
      axi4_vif.ARSIZE  = req.SIZE;
      axi4_vif.ARVALID = 1'b1;

      // Wait for address acceptance
      repeat (20) @(negedge axi4_vif.ACLK)
        if (axi4_vif.ARREADY) break;
      axi4_vif.ARVALID = 0;

      //`uvm_info(get_type_name(), "Read address phase completed", UVM_MEDIUM)
```

```
    end else
    if (req.OPERATION == 2'd1)
    begin

        // data randomiz
        for (int i = 0; i <= req.LEN; i++)
          begin
            assert(req.randomize(DATA))
              else `uvm_fatal(get_type_name(), $sformatf("Data randomization failed for beat %0d", i))
            req.all_data.push_back(req.DATA);

          end

        // Apply delay before starting write address phase
        repeat (req.aw_valid_delay) @(negedge axi4_vif.ACLK);
        // Start write address transaction

        axi4_vif.AWADDR  = req.ADDR;
        axi4_vif.AWLEN   = req.LEN;
        axi4_vif.AWSIZE  = req.SIZE;
        axi4_vif.AWVALID = 1'b1;

        // Wait for address acceptance
        repeat (20) @(negedge axi4_vif.ACLK)
          if (axi4_vif.AWREADY) break;

        axi4_vif.AWVALID = 1'b0;

        //`uvm_info(get_type_name(), "Write address phase completed", UVM_MEDIUM)
        //req.display();

        if (((req.ADDR >> req.SIZE) + (req.LEN + 1)) > 1024)
        begin
            assert(req.randomize(w_valid_delay))
              else $fatal("w_valid_delay randomization failed");

            repeat (req.w_valid_delay) @(negedge axi4_vif.ACLK);

            axi4_vif.WDATA  = req.all_data[0];
            axi4_vif.WVALID = 1'b1;
            axi4_vif.WLAST  = 1;

            // Wait for write data acceptance
            repeat (20) @(negedge axi4_vif.ACLK)
            if (axi4_vif.WREADY) break;

        end else
        begin
```

```
            end else
            begin

                // Apply delay before starting write data phase
                repeat (req.w_valid_delay) @(negedge axi4_vif.ACLK);

                // Send burst data with proper handshaking
                for (int i = 0; i <= req.LEN; i++)
                begin
                    axi4_vif.WDATA  = req.all_data[i];
                    axi4_vif.WVALID = 1'b1;
                    axi4_vif.WLAST  = (i == req.LEN);

                    // Wait for write data acceptance
                    repeat (20) @(negedge axi4_vif.ACLK)
                      if (axi4_vif.WREADY) break;

                    // Generate random delay for each write data beat
                    if (i < req.LEN)
                    begin
                        assert(req.randomize(w_valid_delay))
                            else $fatal("w_valid_delay randomization failed");
                        axi4_vif.WVALID = 1'b0;
                        repeat (req.w_valid_delay) @(negedge axi4_vif.ACLK);
                    end;
                end
                //`uvm_info(get_type_name(), "Write data phase completed", UVM_MEDIUM)
                //req.display();
            end
        end
        //req.print();

        // Store current transaction in config for monitor access
        m_cfg.current_tr = req;
        -> m_cfg.stimulus_sent_e;
    endtask



    `endif
```

AXI4_Monitor:

```
class axi4_monitor extends uvm_monitor;

    `uvm_component_utils(axi4_monitor)

    static int monitor_transaction_count = 0;

    virtual axi4_if axi4_vif;
    common_cfg m_cfg;

    uvm_analysis_port #(axi4_transaction) ap;

    function new (string name = "axi4_monitor", uvm_component parent = null);
        super.new(name,parent);
        ap = new("ap", this);
        `uvm_info("axi4_monitor", "INSIDE NEW MONITOR CLASS", UVM_LOW)
    endfunction

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        `uvm_info(get_type_name(), "axi4 monitor build phase", UVM_LOW)

        if(!uvm_config_db#(virtual axi4_if)::get(this, "*", "axi4_intf", axi4_vif ))
            `uvm_fatal(get_full_name(), "Faild to get axi4_vifface")
        else
            $display("configuration DB dond");

        if(!uvm_config_db#(common_cfg)::get(this, "*", "m_cfg", m_cfg))
            `uvm_fatal(get_full_name(), "Failed to get common_cfg from config DB")
        else
            $display("common_cfg retrieved successfully in monitor");
    endfunction

    task run_phase(uvm_phase phase);
        axi4_transaction tr;

        //`uvm_info(get_type_name(), "ENTERED RUN PHASE", UVM_LOW)

        forever
        begin
            // Wait for trulus to be sent
            wait(m_cfg.stimulus_sent_e.triggered);

            /*monitor_transaction_count++;
            `uvm_info(get_type_name(), $sformatf("Monitor processing transaction %0d", monitor_transaction_count), UVM_LOW)*/

            // Get transaction information from driver via config
            tr = m_cfg.current_tr;

            tr.actual_queue.delete();
            tr.actual_response = 0;
```

```
            if (tr.OPERATION == 2'd2)
            begin

                // Start with RREADY high, then apply delay by temporarily deasserting
                axi4_vif.RREADY = 1'b1;

                axi4_vif.RREADY = 1'b1;

                for (int i = 0; i <= tr.LEN; i++)
                begin
                    tr.actual_response = 2'b00;

                    // Generate random delay for each read beat
                    assert(tr.randomize(r_ready_delay))
                        else $fatal("r_ready_delay randomization failed");

                    // Apply r_ready_delay for each read beat
                    if (tr.r_ready_delay > 0)
                    begin
                      axi4_vif.RREADY = 1'b0;
                      repeat (tr.r_ready_delay) @(negedge axi4_vif.ACLK);
                      axi4_vif.RREADY = 1'b1;
                    end


                    if (i == tr.LEN && !axi4_vif.RLAST)
                    begin
                        `uvm_error(get_type_name(), "RLAST not asserted on final beat")
                    end

                    // Wait for valid read data
                    repeat (20) @(negedge axi4_vif.ACLK)
                      if (axi4_vif.RVALID) break;

                    tr.actual_queue.push_back(axi4_vif.RDATA);
                    tr.actual_response = axi4_vif.RRESP;

                end
                axi4_vif.RREADY = 1'b0;
                //`uvm_info(get_type_name(), "Read operation completed", UVM_MEDIUM)
```

```
            end else
            if (tr.OPERATION == 2'd1)
            begin
                // Generate random delay for each read beat
                assert(tr.randomize(b_ready_delay))
                    else $fatal("b_ready_delay randomization failed");

                tr.actual_response = axi4_vif.BRESP;
                axi4_vif.WVALID = 1'b0;
                axi4_vif.WLAST  = 0;

                // Apply delay before accepting write response
                repeat (tr.b_ready_delay) @(negedge axi4_vif.ACLK);
                axi4_vif.BREADY = 1'b1;

                // Wait for write response
                repeat (20) @(negedge axi4_vif.ACLK)
                  if (axi4_vif.BVALID) break;

                axi4_vif.BREADY = 0;
                //`uvm_info(get_type_name(), "Write operation completed", UVM_MEDIUM)
            end

            // Send transaction to scoreboard and coverage via analysis port
            ap.write(tr);
            -> m_cfg.monitor_sent_e;
        end
    endtask



endclass
`endif
```

AXI4_Sequencer:

```
`ifndef AXI4_SEQUENCER_SVH
`define AXI4_SEQUENCER_SVH

`include "uvm_macros.svh"
import uvm_pkg::*;

`include "axi4_transaction.sv"
`include "common_cfg.sv"


class axi4_sequencer extends uvm_sequencer #(axi4_transaction);

    `uvm_component_utils(axi4_sequencer)

    common_cfg m_cfg;

    function new (string name = "axi4_sequencer", uvm_component parent = null);
        super.new(name,parent);
        `uvm_info("axi4_drive", "INSIDE NEW DRIVER CLASS", UVM_LOW)
    endfunction

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);

        if(!uvm_config_db#(common_cfg)::get(this, "*", "m_cfg", m_cfg))
            `uvm_fatal(get_full_name(), "Failed to get common_cfg from config DB")
        else
            $display("common_cfg retrieved successfully in sequencer");

        `uvm_info(get_type_name(), "axi4 sequencer build phase", UVM_LOW)
    endfunction

endclass
`endif
```

AXI4_Agent:

```
class axi4_agent extends uvm_agent;

    axi4_sequencer sqr;
    axi4_driver    drv;
    axi4_monitor   mon;

    uvm_active_passive_enum is_active = UVM_ACTIVE;


    `uvm_component_utils(axi4_agent)

    function  new(string name = "axi4_agent", uvm_component parent = null);
        super.new(name,parent);
        `uvm_info("axi4_agent", "INSIDE NEW AGENT CLASS", UVM_LOW)
    endfunction


    function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        `uvm_info(get_type_name(), "axi4 agent build phase", UVM_LOW)

        if(!uvm_config_db #(uvm_active_passive_enum)::get(null, "uvm_test_top.env.agt", "is_active", is_active))
            `uvm_fatal(get_type_name(), "Failed to get agent enum value...")
        else
            $display("configuration DB done");

        `uvm_info(get_type_name(), $sformatf("AGENT TYPE IS %p", is_active), UVM_LOW)


        mon = axi4_monitor::type_id::create("mon", this);

        if (is_active == UVM_ACTIVE)
        begin
            sqr = axi4_sequencer::type_id::create("sqr", this);
            drv = axi4_driver::type_id::create("drv", this);
        end
    endfunction


    function void connect_phase(uvm_phase phase);
        super.connect_phase(phase);
        if (is_active == UVM_ACTIVE)
        begin
            drv.seq_item_port.connect(sqr.seq_item_export);
            `uvm_info("my_axi4_agent", "INSIDE CONNECT PHASE", UVM_LOW)
        end
    endfunction

endclass

`endif
```

AXI4_Coverage:

```
class axi4_coverage extends uvm_component;

    `uvm_component_utils(axi4_coverage)

    uvm_analysis_export #(axi4_transaction) analysis_export;
    uvm_tlm_analysis_fifo #(axi4_transaction) fifo;

    axi4_transaction tr;

    covergroup axi4_cov(ref axi4_transaction tr);
        // LEN coverage with corner bins
        coverpoint tr.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 tr.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 tr.SIZE {
          bins fixed_size = {2};
          illegal_bins others = default;
        }
```

```
        // DATA coverage with corner bins
        coverpoint tr.DATA {
          bins corner0       = {32'd0};
          bins corner1       = {32'd1};
          bins corner2       = {32'hFFFF_FFFF};
          bins corner3       = {32'hAAAA_AAAA};
          bins corner4       = {32'h1111_0000};
          bins corner5       = {32'h0000_1111};
          bins range_0       = { [32'd0          : 32'd255] };
          bins range_1       = { [32'd256        : 32'd1023] };
          bins range_2       = { [32'd1024       : 32'd4095] };
          bins range_3       = { [32'd4096       : 32'd16383] };
          bins range_4       = { [32'd16384      : 32'd32767] };
          bins range_5       = { [32'd32768      : 32'd49151] };
          bins range_6       = { [32'd49152      : 32'd65535] };
          bins range_7       = { [32'd65536      : 32'd262143] };
          bins range_8       = { [32'd262144     : 32'd1048575] };
          bins range_9       = { [32'd1048576    : 32'd16777215] };
          bins range_10      = { [32'd16777216   : 32'd268435455] };
          bins range_11      = { [32'd268435456  : 32'd1073741823] };
          bins range_12      = { [32'd1073741824 : 32'hFFFFFFFF] };
          }

        // Memory access bounds coverage
        coverpoint ((tr.ADDR >> 2) + (tr.LEN + 1)) {
          bins valid_access   = {[0:1024]};   // Fits within memory
          bins invalid_access = {[1025:$]};   // Exceeds memory
        }

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

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

    endgroup
```

```
    function new (string name = "axi4_coverage", uvm_component parent = null);
        super.new(name,parent);
        axi4_cov = new(tr);

        `uvm_info("axi4_coverage", "INSIDE NEW COVERAGE CLASS", UVM_LOW)
    endfunction

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        analysis_export = new("analysis_export", this);
        fifo = new("fifo", this);

        `uvm_info(get_type_name(), "axi4 coverage build phase", UVM_LOW)
    endfunction

    function void connect_phase(uvm_phase phase);
        analysis_export.connect(fifo.analysis_export);
    endfunction

    task run_phase(uvm_phase phase);
        forever
        begin
            fifo.get(tr);
            axi4_cov.sample();
            //`uvm_info(get_type_name(), "Coverage sampled for transaction", UVM_MEDIUM)
        end
    endtask

    function void report_phase(uvm_phase phase);
      real cov = axi4_cov.get_coverage();
      `uvm_info(get_type_name(),
                $sformatf("Final Coverage = %0.2f%%", cov),
                UVM_LOW)
    endfunction

endclass
`endif
```

AXI4_Scoreboard:

- Golden model

```
    wait(m_cfg.monitor_sent_e.triggered);

    fifo.get(tr);
    cases++;

    // Debug: Print every transaction received
    /*`uvm_info(get_type_name(),
        $sformatf("Scoreboard received transaction %0d: OPERATION = %0d (%s)",
        cases,
        tr.OPERATION,
        (tr.OPERATION == 2'd1) ? "WRITE" : "READ"),UVM_LOW)*/

    expected_queue.delete();
    expected_response = 0;

    // Golden model
    if (((tr.ADDR >> tr.SIZE) + (tr.LEN)) >= 1024)
    begin
        expected_response = 2'b10;
    end else
    begin
        expected_response = 2'b00;
        if (tr.OPERATION == 2'd1)
        begin
            // Write operation - update golden memory
            for (int i = 0; i <= tr.LEN; i++)
            begin
                golden_mem[(tr.ADDR >> 2) + i] = tr.all_data[i];
            end
        end else if (tr.OPERATION == 2'd2)
        begin
            // Read operation - get expected data from golden memory
            for (int i = 0; i <= tr.LEN; i++)
            begin
                expected_queue.push_back(golden_mem[(tr.ADDR >> 2) + i]);
            end
        end
    end
```

- Check:

```
                // Check results
                if (tr.OPERATION == 2'd1)
                begin
                    `uvm_info(get_type_name(), $sformatf("Test %0d - Write Operation", cases), UVM_LOW)
                    tr.display();

                    if (tr.actual_response == expected_response)
                    begin
                        pass++;
                        `uvm_info("get_type_name()", $sformatf("Test %0d PASSED", cases), UVM_LOW)
                        `uvm_info("get_type_name()", "Write Operation passed", UVM_LOW)
                        `uvm_info("get_type_name()", $sformatf("Expected response: %2b, Actual response: %2b",
                                                               expected_response, tr.actual_response), UVM_LOW)
                    end
                    else
                    begin
                        fail++;
                        faild_cases.push_back(cases);
                        `uvm_info("get_type_name()", $sformatf("Test %0d FAILED - Response mismatch", cases), UVM_LOW)
                        `uvm_info("get_type_name()", "Write Operation failed", UVM_LOW)
                        `uvm_info("get_type_name()", $sformatf("Expected response: %2b, Actual response: %2b",
                                                               expected_response, tr.actual_response), UVM_LOW)
                    end

                end else
                if (tr.OPERATION == 2'd2)
                begin
                    `uvm_info(get_type_name(), $sformatf("Test %0d - Read Operation", cases), UVM_LOW)
                    tr.display();

                    if (((tr.ADDR >> tr.SIZE) + (tr.LEN)) >= 1024)
                    begin
                        if (tr.actual_response == expected_response)
                        begin
                            pass++;
                            `uvm_info("get_type_name()", $sformatf("Test %0d PASSED", cases), UVM_LOW)
                            `uvm_info("get_type_name()", "Read Operation passed", UVM_LOW)
                            `uvm_info("get_type_name()", $sformatf("Expected response: %2b, Actual response: %2b",
                                                               expected_response, tr.actual_response), UVM_LOW)
                        end else
                        begin
                            fail++;
                            faild_cases.push_back(cases);
                            `uvm_info("get_type_name()", $sformatf("Test %0d FAILED - Response mismatch", cases), UVM_LOW)
                            `uvm_info("get_type_name()", "Read Operation failed", UVM_LOW)
                            `uvm_info("get_type_name()", $sformatf("Expected response: %2b, Actual response: %2b",
                                                               expected_response, tr.actual_response), UVM_LOW)
                        end
                    end else
                    begin
```

```
                    if (tr.actual_queue.size() != expected_queue.size()) begin
                        fail++;
                        faild_cases.push_back(cases);
                        `uvm_info("get_type_name()", $sformatf("Test %0d FAILED - Queue size mismatch", cases), UVM_LOW)
                        `uvm_info("get_type_name()", "Read Operation failed", UVM_LOW)
                        `uvm_info("get_type_name()", $sformatf("Expected size: %0d, Actual size: %0d",
                                                               expected_queue.size(), tr.actual_queue.size()), UVM_LOW)
                    end else
                    if ((tr.actual_queue == expected_queue) && (tr.actual_response == expected_response))
                    begin
                      pass++;
                      `uvm_info("get_type_name()", $sformatf("Test %0d PASSED", cases), UVM_LOW)
                      `uvm_info("get_type_name()", "Read Operation passed", UVM_LOW)
                      `uvm_info("get_type_name()", $sformatf("Expected response: %2b, Actual response: %2b",
                                                               expected_response, tr.actual_response), UVM_LOW)
                    end else
                    begin
                        fail++;
                        faild_cases.push_back(cases);
                        `uvm_error("get_type_name()", $sformatf("Test %0d FAILED - Data|Response mismatch", cases))
                        `uvm_info("get_type_name()", "Read Operation failed", UVM_LOW)
                        `uvm_info("get_type_name()", $sformatf("Expected response: %2b, Actual response: %2b",
                                                               expected_response, tr.actual_response), UVM_LOW)
                        // Print detailed mismatch information
                        for (int i = 0; i < expected_queue.size(); i++)
                        begin
                            if (expected_queue[i] != tr.actual_queue[i])
                            begin
                              `uvm_info("get_type_name()", $sformatf("  Beat[%0d]: Expected = %h, Actual = %h",
                                                               i, expected_queue[i], tr.actual_queue[i]), UVM_LOW)
                            end
                        end
                    end
                end
            end
        end
        $display("");
    end
endtask

function void extract_phase(uvm_phase phase);
    super.report_phase(phase);
    $display("=== Final Results ===");
    `uvm_info(get_type_name(), $sformatf("Total tests: %0d", cases), UVM_LOW)
    `uvm_info(get_type_name(), $sformatf("Passed: %0d", pass), UVM_LOW)
    `uvm_info(get_type_name(), $sformatf("Failed: %0d", fail), UVM_LOW)
    if (faild_cases.size() > 0)
    begin
        for (int i = 0; i < faild_cases.size(); i++)
        begin
            `uvm_info(get_type_name(), $sformatf("case nember: %0d failed", faild_cases[i]), UVM_LOW)
        end
    end
    if (cases > 0) begin
        cov = (real'(pass) / real'(cases)) * 100.0;
        `uvm_info(get_type_name(), $sformatf("Pass Rate: %0.2f%%", cov), UVM_LOW)
```

AXI4_ENV:

```
class axi4_env extends uvm_env;

    common_cfg      m_cfg;

    axi4_agent      agt;
    axi4_coverage   cov;
    axi4_scoreboard scb;

    `uvm_component_utils(axi4_env)

    function new(string name = "axi4_env", uvm_component parent = null);
        super.new(name,parent);
        `uvm_info("axi4_env", "INSIDE NEW ENV CLASS", UVM_LOW)
    endfunction

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        agt   = axi4_agent::type_id::create("agt", this);
        cov   = axi4_coverage::type_id::create("cov", this);
        scb = axi4_scoreboard::type_id::create("scb", this);
        `uvm_info(get_type_name(), "axi4 enviroment build phase", UVM_LOW)

        if(!uvm_config_db#(common_cfg)::get(this, "", "m_cfg", m_cfg)) begin
            `uvm_fatal(get_full_name(), "Failed to get common_cfg from config DB")
        end else
        begin
            $display("common_cfg retrieved successfully in environment");
        end

        uvm_config_db#(common_cfg)::set(this, "*", "m_cfg", m_cfg);
    endfunction

    function void connect_phase(uvm_phase phase);
        super.connect_phase(phase);

        // Connect config to components
        agt.drv.m_cfg = m_cfg;
        agt.mon.m_cfg = m_cfg;

        // Connect analysis ports
        agt.drv.ap.connect(cov.analysis_export);
        agt.mon.ap.connect(scb.analysis_export);
        agt.mon.ap.connect(cov.analysis_export);

        `uvm_info(get_type_name(), "axi4_env connect phase", UVM_LOW)
    endfunction

endclass
`endif
```

COMMAN_CFG:

```
`ifndef COMMON_CFG_SVH
`define COMMON_CFG_SVH


`include "uvm_macros.svh"
import uvm_pkg::*;

`include "axi4_transaction.sv"



class common_cfg extends uvm_object;

    `uvm_object_utils(common_cfg)

    event stimulus_sent_e;
    event monitor_sent_e;

    axi4_transaction current_tr;

    function new(string name = "common_cfg");
        super.new(name);
        `uvm_info("common_cfg", "INSIDE NEW COMMON_CFG CLASS", UVM_LOW)
    endfunction

endclass
`endif
```

AXI4_Assertions:

```
module axi4_assert (axi4_if axi4_vif);

  // All outputs should be properly initialized after reset
  property reset_awready;
    @(posedge axi4_vif.ACLK) !axi4_vif.ARESETn |-> axi4_vif.AWREADY == 1'b1;
  endproperty
  A_RESET_AWREADY: assert property (reset_awready)
    else `uvm_error("AXI_ASSERT", "AWREADY not initialized to 1 after reset");
  C_RESET_AWREADY: cover property (reset_awready);

  property reset_wready;
    @(posedge axi4_vif.ACLK) !axi4_vif.ARESETn |-> axi4_vif.WREADY == 1'b0;
  endproperty
  A_RESET_WREADY: assert property (reset_wready)
    else `uvm_error("AXI_ASSERT", "WREADY not initialized to 0 after reset");
  C_RESET_WREADY: cover property (reset_wready);

  property reset_bvalid;
    @(posedge axi4_vif.ACLK) !axi4_vif.ARESETn |-> axi4_vif.BVALID == 1'b0;
  endproperty
  A_RESET_BVALID: assert property (reset_bvalid)
    else `uvm_error("AXI_ASSERT", "BVALID not initialized to 0 after reset");
  C_RESET_BVALID: cover property (reset_bvalid);

  property reset_arready;
    @(posedge axi4_vif.ACLK) !axi4_vif.ARESETn |-> axi4_vif.ARREADY == 1'b1;
  endproperty
  A_RESET_ARREADY: assert property (reset_arready)
    else `uvm_error("AXI_ASSERT", "ARREADY not initialized to 1 after reset");
  C_RESET_ARREADY: cover property (reset_arready);

  property reset_rvalid;
    @(posedge axi4_vif.ACLK) !axi4_vif.ARESETn |-> axi4_vif.RVALID == 1'b0;
  endproperty
  A_RESET_RVALID: assert property (reset_rvalid)
    else `uvm_error("AXI_ASSERT", "RVALID not initialized to 0 after reset");
  C_RESET_RVALID: cover property (reset_rvalid);

  property reset_rlast;
    @(posedge axi4_vif.ACLK) !axi4_vif.ARESETn |-> axi4_vif.RLAST == 1'b0;
  endproperty
  A_RESET_RLAST: assert property (reset_rlast)
    else `uvm_error("AXI_ASSERT", "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 axi4_vif.ACLK) disable iff (!axi4_vif.ARESETn)
    (axi4_vif.AWVALID && axi4_vif.AWREADY) |=> !axi4_vif.AWREADY;
  endproperty
  A_AWREADY_DEASSERT: assert property (awready_deassert)
    else `uvm_error("AXI_ASSERT", "AWREADY should deassert after address handshake");
  C_AWREADY_DEASSERT: cover property (awready_deassert);
```

```

  // Address signals should remain stable when AWVALID is high
  property awaddr_stable;
    @(posedge axi4_vif.ACLK) disable iff (!axi4_vif.ARESETn)
    axi4_vif.AWVALID && !axi4_vif.AWREADY |=> $stable(axi4_vif.AWADDR);
  endproperty
  A_AWADDR_STABLE: assert property (awaddr_stable)
    else `uvm_error("AXI_ASSERT", "AWADDR must remain stable when AWVALID is high");
  C_AWADDR_STABLE: cover property (awaddr_stable);

  property awlen_stable;
    @(posedge axi4_vif.ACLK) disable iff (!axi4_vif.ARESETn)
    axi4_vif.AWVALID && !axi4_vif.AWREADY |=> $stable(axi4_vif.AWLEN);
  endproperty
  A_AWLEN_STABLE: assert property (awlen_stable)
    else `uvm_error("AXI_ASSERT", "AWLEN must remain stable when AWVALID is high");
  C_AWLEN_STABLE: cover property (awlen_stable);

  property awsize_stable;
    @(posedge axi4_vif.ACLK) disable iff (!axi4_vif.ARESETn)
    axi4_vif.AWVALID && !axi4_vif.AWREADY |=> $stable(axi4_vif.AWSIZE);
  endproperty
  A_AWSIZE_STABLE: assert property (awsize_stable)
    else `uvm_error("AXI_ASSERT", "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 axi4_vif.ACLK) disable iff (!axi4_vif.ARESETn)
    axi4_vif.WVALID && !axi4_vif.WREADY |=> $stable(axi4_vif.WDATA);
  endproperty
  A_WDATA_STABLE: assert property (wdata_stable)
    else `uvm_error("AXI_ASSERT", "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 axi4_vif.ACLK) disable iff (!axi4_vif.ARESETn)
    (axi4_vif.WVALID && axi4_vif.WREADY && axi4_vif.WLAST) |=> !axi4_vif.WREADY;
  endproperty
  A_WLAST_LAST_BEAT: assert property (wlast_on_last_beat)
    else `uvm_error("AXI_ASSERT", "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 axi4_vif.ACLK) disable iff (!axi4_vif.ARESETn)
    $rose(axi4_vif.BVALID) |-> $past(axi4_vif.WVALID && axi4_vif.WREADY && axi4_vif.WLAST);
  endproperty
  A_WRITE_ORDER_DATA_RESP: assert property (write_order)
    else `uvm_error("AXI_ASSERT", "Write response cannot start without data completion");
  C_WRITE_ORDER_DATA_RESP: cover property (write_order);
```

```
  // BVALID should be asserted after write data completion
  property bvalid_after_wlast;
    @(posedge axi4_vif.ACLK) disable iff (!axi4_vif.ARESETn)
    (axi4_vif.WVALID && axi4_vif.WREADY && axi4_vif.WLAST) |=> axi4_vif.BVALID;
  endproperty
  A_BVALID_AFTER_WLAST: assert property (bvalid_after_wlast)
    else `uvm_error("AXI_ASSERT", "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 axi4_vif.ACLK) disable iff (!axi4_vif.ARESETn)
    axi4_vif.BVALID && !axi4_vif.BREADY |=> axi4_vif.BVALID;
  endproperty
  A_BVALID_STABLE: assert property (bvalid_stable)
    else `uvm_error("AXI_ASSERT", "BVALID must remain stable until handshake");
  C_BVALID_STABLE: cover property (bvalid_stable);

  // BRESP should remain stable when BVALID is high
  property bresp_stable;
    @(posedge axi4_vif.ACLK) disable iff (!axi4_vif.ARESETn)
    axi4_vif.BVALID && !axi4_vif.BREADY |=> $stable(axi4_vif.BRESP);
  endproperty
  A_BRESP_STABLE: assert property (bresp_stable)
    else `uvm_error("AXI_ASSERT", "BRESP must remain stable when BVALID is high");
  C_BRESP_STABLE: cover property (bresp_stable);

  // BVALID should deassert after handshake
  property bvalid_deassert;
    @(posedge axi4_vif.ACLK) disable iff (!axi4_vif.ARESETn)
    (axi4_vif.BVALID && axi4_vif.BREADY) |=> !axi4_vif.BVALID;
  endproperty
  A_BVALID_DEASSERT: assert property (bvalid_deassert)
    else `uvm_error("AXI_ASSERT", "BVALID should deassert after response handshake");
  C_BVALID_DEASSERT: cover property (bvalid_deassert);

  // ARREADY should go low after accepting address
  property arready_deassert;
    @(posedge axi4_vif.ACLK) disable iff (!axi4_vif.ARESETn)
    (axi4_vif.ARVALID && axi4_vif.ARREADY) |=> !axi4_vif.ARREADY;
  endproperty
  A_ARREADY_DEASSERT: assert property (arready_deassert)
    else `uvm_error("AXI_ASSERT", "ARREADY should deassert after address handshake");
  C_ARREADY_DEASSERT: cover property (arready_deassert);

  // Read address signals should remain stable when ARVALID is high
  property araddr_stable;
    @(posedge axi4_vif.ACLK) disable iff (!axi4_vif.ARESETn)
    axi4_vif.ARVALID && !axi4_vif.ARREADY |=> $stable(axi4_vif.ARADDR);
  endproperty
  A_ARADDR_STABLE: assert property (araddr_stable)
    else `uvm_error("AXI_ASSERT", "ARADDR must remain stable when ARVALID is high");
  C_ARADDR_STABLE: cover property (araddr_stable);
```

```
  property arlen_stable;
    @(posedge axi4_vif.ACLK) disable iff (!axi4_vif.ARESETn)
    axi4_vif.ARVALID && !axi4_vif.ARREADY |=> $stable(axi4_vif.ARLEN);
  endproperty
  A_ARLEN_STABLE: assert property (arlen_stable)
    else `uvm_error("AXI_ASSERT", "ARLEN must remain stable when ARVALID is high");
  C_ARLEN_STABLE: cover property (arlen_stable);

  property arsize_stable;
    @(posedge axi4_vif.ACLK) disable iff (!axi4_vif.ARESETn)
    axi4_vif.ARVALID && !axi4_vif.ARREADY |=> $stable(axi4_vif.ARSIZE);
  endproperty
  A_ARSIZE_STABLE: assert property (arsize_stable)
    else `uvm_error("AXI_ASSERT", "ARSIZE must remain stable when ARVALID is high");
  C_ARSIZE_STABLE: cover property (arsize_stable);


  // RVALID should be asserted after read address
  property rvalid_after_araddr;
    @(posedge axi4_vif.ACLK) disable iff (!axi4_vif.ARESETn)
    (axi4_vif.ARVALID && axi4_vif.ARREADY) |-> ##[1:3] axi4_vif.RVALID;
  endproperty
  A_RVALID_AFTER_ARADDR: assert property (rvalid_after_araddr)
    else `uvm_error("AXI_ASSERT", "RVALID should be asserted within 3 cycles after read address");
  C_RVALID_AFTER_ARADDR: cover property (rvalid_after_araddr);

  // RVALID should remain stable until RREADY
  property rvalid_stable;
    @(posedge axi4_vif.ACLK) disable iff (!axi4_vif.ARESETn)
    axi4_vif.RVALID && !axi4_vif.RREADY |=> axi4_vif.RVALID;
  endproperty
  A_RVALID_STABLE: assert property (rvalid_stable)
    else `uvm_error("AXI_ASSERT", "RVALID must remain stable until handshake");
  C_RVALID_STABLE: cover property (rvalid_stable);

  // RRESP should remain stable when RVALID is high
  property rresp_stable;
    @(posedge axi4_vif.ACLK) disable iff (!axi4_vif.ARESETn)
    axi4_vif.RVALID && !axi4_vif.RREADY |=> $stable(axi4_vif.RRESP);
  endproperty
  A_RRESP_STABLE: assert property (rresp_stable)
    else `uvm_error("AXI_ASSERT", "RRESP must remain stable when RVALID is high");
  C_RRESP_STABLE: cover property (rresp_stable);

  // RLAST should remain stable when RVALID is high
  property rlast_stable;
    @(posedge axi4_vif.ACLK) disable iff (!axi4_vif.ARESETn)
    axi4_vif.RVALID && !axi4_vif.RREADY |=> $stable(axi4_vif.RLAST);
  endproperty
  A_RLAST_STABLE: assert property (rlast_stable)
    else `uvm_error("AXI_ASSERT", "RLAST must remain stable when RVALID is high");
  C_RLAST_STABLE: cover property (rlast_stable);
```

```
231   // RRESP should be OKAY (00) or SLVERR (10)
232   property rresp_valid_values;
233     @(posedge axi4_vif.ACLK) disable iff (!axi4_vif.ARESETn)
234     axi4_vif.RVALID |-> (axi4_vif.RRESP == 2'b00 || axi4_vif.RRESP == 2'b10);
235   endproperty
236   A_RRESP_VALID_VALUES: assert property (rresp_valid_values)
237     else `uvm_error("AXI_ASSERT", $sformatf("Invalid RRESP value: %b", axi4_vif.RRESP));
238   C_RRESP_VALID_VALUES: cover property (rresp_valid_values);
239
240   // 4KB boundary crossing should result in SLVERR for write
241   property write_boundary;
242     @(posedge axi4_vif.ACLK) disable iff (!axi4_vif.ARESETn)
243     (axi4_vif.BVALID &&
244      (((axi4_vif.AWADDR & 16'h0FFF) + ((axi4_vif.AWLEN) << axi4_vif.AWSIZE)) > 16'h0FFF))
245     |-> axi4_vif.BRESP == 2'b10;
246   endproperty
247   A_WRITE_BOUNDARY_ERROR: assert property (write_boundary)
248     else `uvm_error("AXI_ASSERT", "4KB boundary crossing should result in SLVERR");
249   C_WRITE_BOUNDARY_ERROR: cover property (write_boundary);
250
251   // Out of memory range should result in SLVERR for write
252   property write_range;
253     @(posedge axi4_vif.ACLK) disable iff (!axi4_vif.ARESETn)
254     (axi4_vif.BVALID && ((axi4_vif.AWADDR >> 2) >= 1024))
255     |-> axi4_vif.BRESP == 2'b10;
256   endproperty
257   A_WRITE_RANGE_ERROR: assert property (write_range)
258     else `uvm_error("AXI_ASSERT", "Out of range write should result in SLVERR");
259   C_WRITE_RANGE_ERROR: cover property (write_range);
260
261   // 4KB boundary crossing should result in SLVERR for read
262   property read_boundary;
263     @(posedge axi4_vif.ACLK) disable iff (!axi4_vif.ARESETn)
264     (axi4_vif.RVALID &&
265      (((axi4_vif.ARADDR & 16'h0FFF) + ((axi4_vif.ARLEN) << axi4_vif.ARSIZE)) > 16'h0FFF))
266     |-> axi4_vif.RRESP == 2'b10;
267   endproperty
268   A_READ_BOUNDARY_ERROR: assert property (read_boundary)
269     else `uvm_error("AXI_ASSERT", "4KB boundary crossing should result in SLVERR for read");
270   C_READ_BOUNDARY_ERROR: cover property (read_boundary);
271
272   // Out of memory range should result in SLVERR for read
273   property read_range;
274     @(posedge axi4_vif.ACLK) disable iff (!axi4_vif.ARESETn)
275     (axi4_vif.RVALID && ((axi4_vif.ARADDR >> 2) >= 1024))
276     |-> axi4_vif.RRESP == 2'b10;
277   endproperty
278   A_READ_RANGE_ERROR: assert property (read_range)
279     else `uvm_error("AXI_ASSERT", "Out of range read should result in SLVERR");
280   C_READ_RANGE_ERROR: cover property (read_range);
```

System Results:

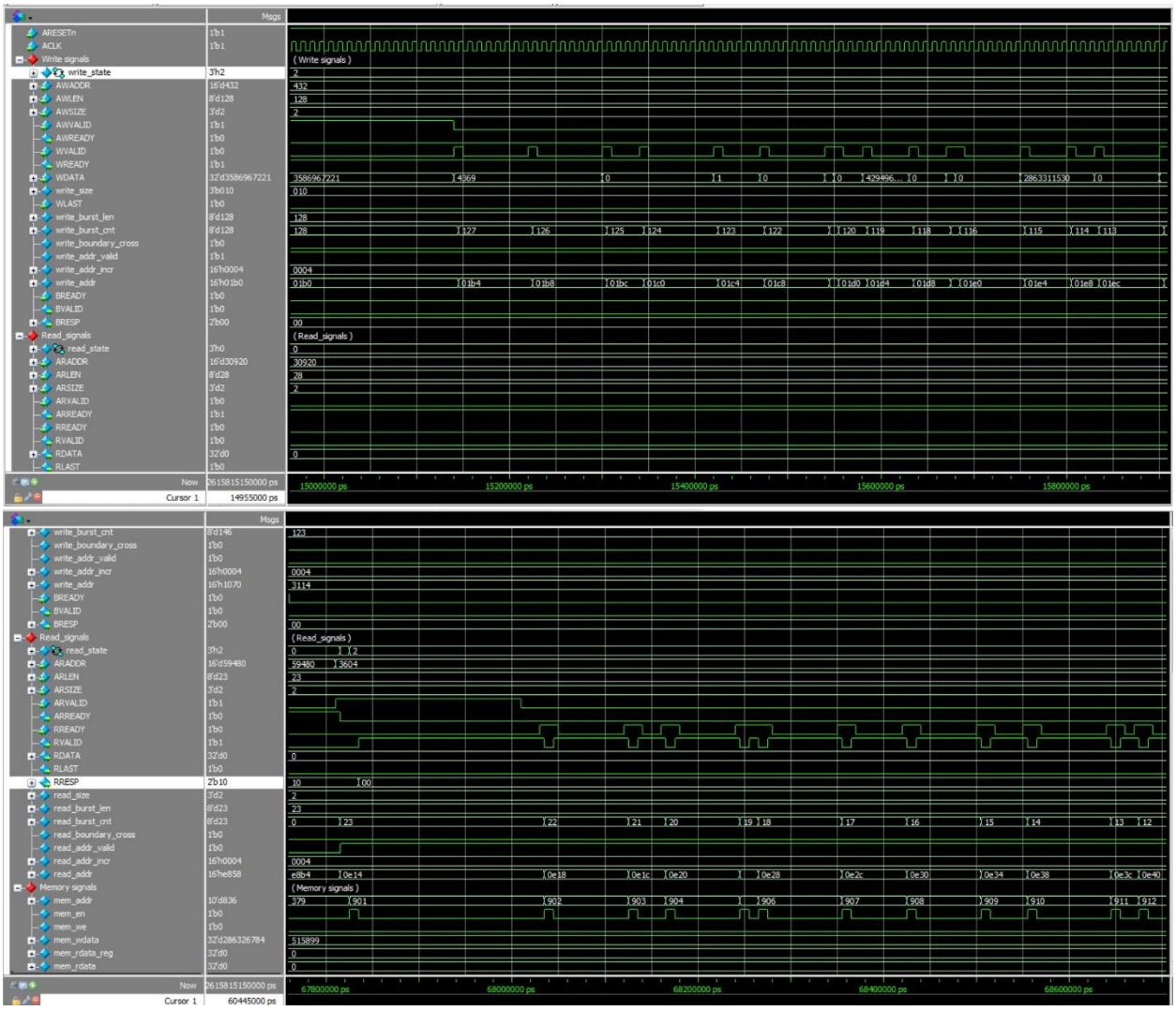
- Transcript:

```
#
# UVM_INFO axi4_sequence.sv(63) @ 2614447530000: reporter@@axi4_sequence [axi4_sequence] Sequence completed - Generated 500000 transactions
# UVM_INFO axi4_test.sv(53) @ 2614447530000: uvm_test_top [axi4_test] AXI4 test sequence completed
# UVM_INFO debug_sequence.sv(53) @ 2614447530000: reporter@@debug_sequence [debug_sequence] Starting debug sequence with targeted scenarios
# UVM_INFO debug_sequence.sv(57) @ 2614447530000: reporter@@debug_sequence [debug_sequence] Testing scenario 0: Read near 4KB boundary
# UVM_INFO axi4_transaction.sv(204) @ 2614447530000: reporter@@axi4_transaction [axi4_transaction] INSIDE NEW TRANSACTION CLASS
# UVM_INFO axi4_scoreboard.sv(137) @ 2614448110000: uvm_test_top.env.scb [axi4_scoreboard] Test 500001 - Read Operation
# UVM_INFO axi4_transaction.sv(177) @ 2614448110000: reporter@@axi4_transaction [axi4_transaction] ADDR = 4080, LEN = 3, SIZE = 2, Memory Access = 1023
# UVM_INFO axi4_transaction.sv(180) @ 2614448110000: reporter@@axi4_transaction [axi4_transaction] Delays - AW:3, W:0, B:6, AR:1, R:2
# UVM_INFO axi4_scoreboard.sv(173) @ 2614448110000: uvm_test_top.env.scb [get_type_name()] Test 500001 PASSED
# UVM_INFO axi4_scoreboard.sv(174) @ 2614448110000: uvm_test_top.env.scb [get_type_name()] Read Operation passed
# UVM_INFO axi4_scoreboard.sv(175) @ 2614448110000: uvm_test_top.env.scb [get_type_name()] Expected response: 00, Actual response: 00
#
```

```
#
# UVM_INFO debug_sequence.sv(104) @ 2615815150000: reporter@@debug_sequence [debug_sequence] Completed scenario 9: Large read in middle range
# UVM_INFO debug_sequence.sv(109) @ 2615815150000: reporter@@debug_sequence [debug_sequence] Debug sequence completed - Generated 500 targeted transactions
# UVM_INFO axi4_test.sv(56) @ 2615815150000: uvm_test_top [axi4_test] Debug sequence completed
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 2615815150000: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# === Final Results ===
# UVM_INFO axi4_scoreboard.sv(206) @ 2615815150000: uvm_test_top.env.scb [axi4_scoreboard] Total tests: 500500
# UVM_INFO axi4_scoreboard.sv(207) @ 2615815150000: uvm_test_top.env.scb [axi4_scoreboard] Passed: 500500
# UVM_INFO axi4_scoreboard.sv(208) @ 2615815150000: uvm_test_top.env.scb [axi4_scoreboard] Failed: 0
# UVM_INFO axi4_scoreboard.sv(218) @ 2615815150000: uvm_test_top.env.scb [axi4_scoreboard] Pass Rate: 100.00%
# UVM_INFO axi4_coverage.sv(128) @ 2615815150000: uvm_test_top.env.cov [axi4_coverage] Final Coverage = 100.00%
```

```
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :3503558
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# [UVMTOP]     1
# [axi4_agent]     3
# [axi4_coverage]     3
# [axi4_drive]     1
# [axi4_driver]     2
# [axi4_env]     3
# [axi4_monitor]     2
# [axi4_scoreboard] 500506
# [axi4_sequence]     2
# [axi4_sequencer]     1
# [axi4_test]     5
# [axi4_transaction] 1501500
# [common_cfg]     1
# [debug_sequence]    23
# [get_type_name()] 1501500
# [my_axi4_agent]     1
# ** Note: $finish    : C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 2615815150 ns  Iteration: 61  Instance: /top
# 1
```

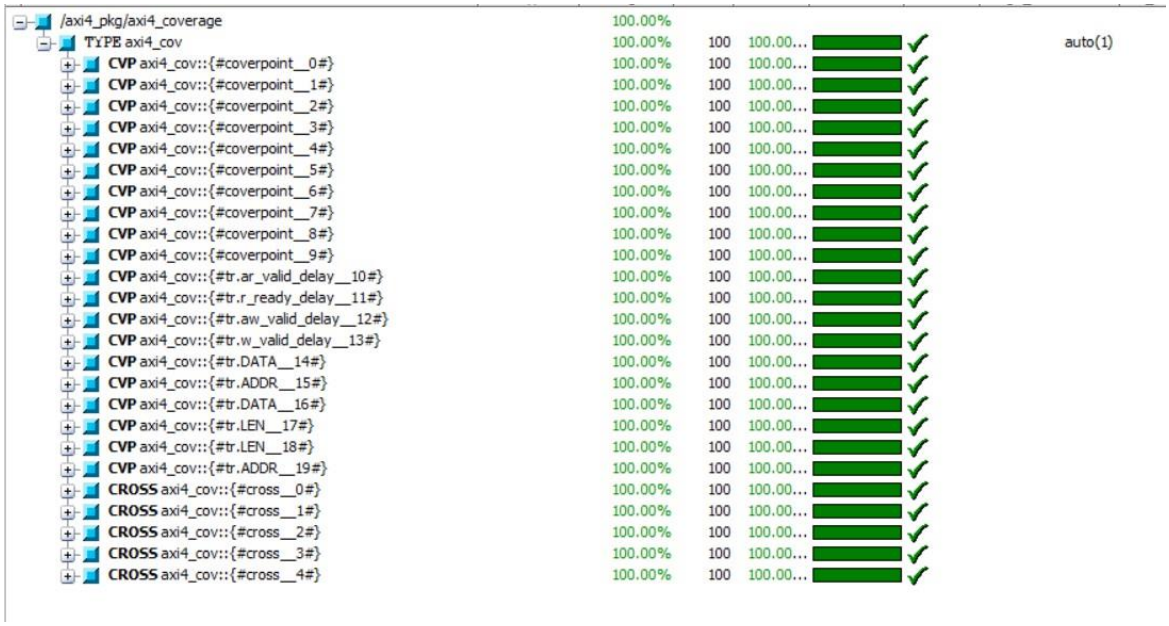
- Wave:



- Assertions:

/top/check/C_RESET_AWREADY	SVA	✓	Off	3	1	Unli...	1	100%	✓	0	80	5000 ps	3
/top/check/C_RESET_WREADY	SVA	✓	Off	3	1	Unli...	1	100%	✓	0	80	5000 ps	3
/top/check/C_RESET_BVALID	SVA	✓	Off	3	1	Unli...	1	100%	✓	0	80	5000 ps	3
/top/check/C_RESET_ARREADY	SVA	✓	Off	3	1	Unli...	1	100%	✓	0	80	5000 ps	3
/top/check/C_RESET_RVALID	SVA	✓	Off	3	1	Unli...	1	100%	✓	0	80	5000 ps	3
/top/check/C_RESET_RLAST	SVA	✓	Off	3	1	Unli...	1	100%	✓	0	80	5000 ps	3
/top/check/C_AWREADY_DEASSERT	SVA	✓	Off	402	1	Unli...	1	100%	✓	0	80	14165000 ps	402
/top/check/C_AWADDR_STABLE	SVA	✓	Off	7638	1	Unli...	1	100%	✓	0	160	14185000 ps	7638
/top/check/C_AWLEN_STABLE	SVA	✓	Off	7638	1	Unli...	1	100%	✓	0	160	14185000 ps	7638
/top/check/C_AWSIZE_STABLE	SVA	✓	Off	7638	1	Unli...	1	100%	✓	0	160	14185000 ps	7638
/top/check/C_WDATA_STABLE	SVA	✓	Off	7638	1	Unli...	1	100%	✓	0	160	14435000 ps	7638
/top/check/C_WLAST_LAST_BEAT	SVA	✓	Off	402	1	Unli...	1	100%	✓	0	80	14415000 ps	402
/top/check/C_WRITE_ORDER_DATA_RES...	SVA	✓	Off	402	1	Unli...	1	100%	✓	0	80	14425000 ps	402
/top/check/C_BVALID_AFTER_WLAST	SVA	✓	Off	402	1	Unli...	1	100%	✓	0	80	14415000 ps	402
/top/check/C_BVALID_STABLE	SVA	✓	Off	9013	1	Unli...	1	100%	✓	0	160	14435000 ps	9013
/top/check/C_BRESP_STABLE	SVA	✓	Off	9013	1	Unli...	1	100%	✓	0	160	14435000 ps	9013
/top/check/C_BVALID_DEASSERT	SVA	✓	Off	402	1	Unli...	1	100%	✓	0	80	14685000 ps	402
/top/check/C_ARREADY_DEASSERT	SVA	✓	Off	598	1	Unli...	1	100%	✓	0	80	75000 ps	598
/top/check/C_ARADDR_STABLE	SVA	✓	Off	11362	1	Unli...	1	100%	✓	0	160	95000 ps	11362
/top/check/C_ARLEN_STABLE	SVA	✓	Off	11362	1	Unli...	1	100%	✓	0	160	95000 ps	11362
/top/check/C_ARSIZE_STABLE	SVA	✓	Off	11362	1	Unli...	1	100%	✓	0	160	95000 ps	11362
/top/check/C_RVALID_AFTER_ARADDR	SVA	✓	Off	598	1	Unli...	1	100%	✓	0	80	75000 ps	598
/top/check/C_RVALID_STABLE	SVA	✓	Off	89654	1	Unli...	1	100%	✓	0	160	115000 ps	189654
/top/check/C_RRESP_STABLE	SVA	✓	Off	89654	1	Unli...	1	100%	✓	0	160	115000 ps	189654
/top/check/C_RLAST_STABLE	SVA	✓	Off	89654	1	Unli...	1	100%	✓	0	160	115000 ps	189654
/top/check/C_BRESP_VALID_VALUES	SVA	✓	Off	9415	1	Unli...	1	100%	✓	0	80	14425000 ps	9415
/top/check/C_RRESP_VALID_VALUES	SVA	✓	Off	40782	1	Unli...	1	100%	✓	0	80	105000 ps	240782
/top/check/C_WRITE_BOUNDARY_ERROR	SVA	✓	Off	603	1	Unli...	1	100%	✓	0	80	21425000 ps	603
/top/check/C_WRITE_RANGE_ERROR	SVA	✓	Off	4830	1	Unli...	1	100%	✓	0	80	14425000 ps	4830
/top/check/C_READ_BOUNDARY_ERROR	SVA	✓	Off	85058	1	Unli...	1	100%	✓	0	80	31155000 ps	85058
/top/check/C_READ_RANGE_ERROR	SVA	✓	Off	19081	1	Unli...	1	100%	✓	0	80	12045000 ps	119081

- Function coverage



- Code coverage:

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

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

FSM Coverage for instance /top/dut --

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

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

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

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

Statement Coverage for instance /top/dut --

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

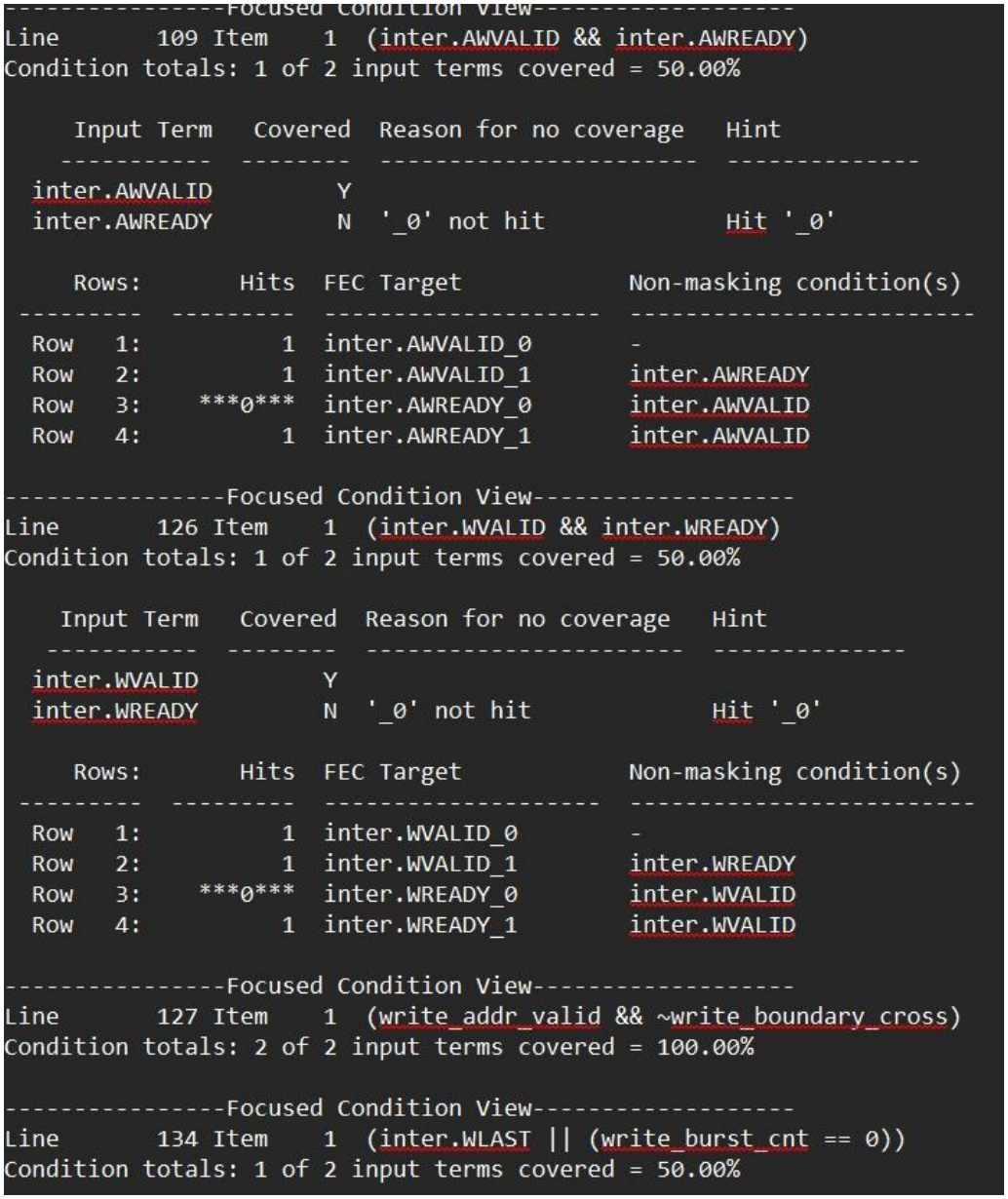
```
Expression Coverage:
    Enabled Coverage              Bins   Covered    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Expressions                      5         5         0   100.00%

================================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)
Expression totals: 1 of 1 input term covered = 100.00%

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

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

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

```
================================================================================
=== Instance: /top/dut
=== Design Unit: work.axi4
================================================================================
Branch Coverage:
    Enabled Coverage              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
                                     AWSIZE[2-0]           0           0        0.00
                                        BRESP[0]           0           0        0.00
                                        RRESP[1]           0           1       50.00
                                        RRESP[0]           0           0        0.00

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

Toggle Coverage      =      90.80% (247 of 272 bins)
```

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

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

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

Condition Coverage for instance /top/dut --

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

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

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

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

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

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



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.

Run.do:

- For coverage:

```
if {![file exists work]} {
   vlib work
}
vlog -f files.txt +cover -covercells
vsim  work.top  -cover
coverage save -onexit cov.ucdb -du work.axi4
do wave.do
run -all
coverage report -details -output cov_report.txt
```

- For Assertions:

```
vlib work
vlog -f files.txt
vsim  -assertdebug +acc -voptargs=+acc work.top
run -all
```