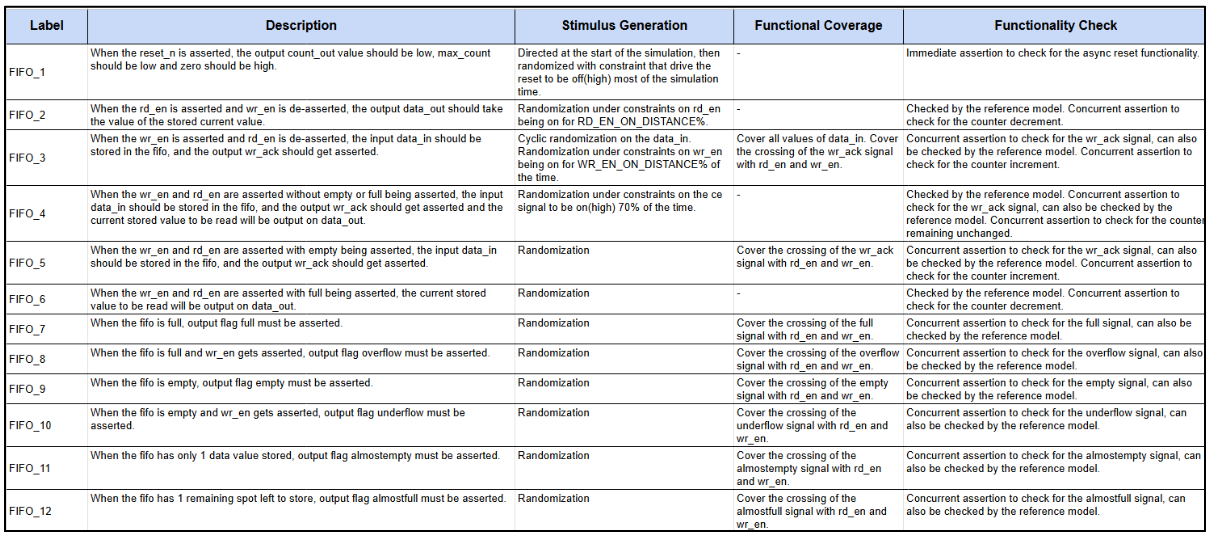
Synchronous FIFO – SV

Amira Atef Ismaeil El Komy

Table of Contents

List of Figures

1. Verification Plan

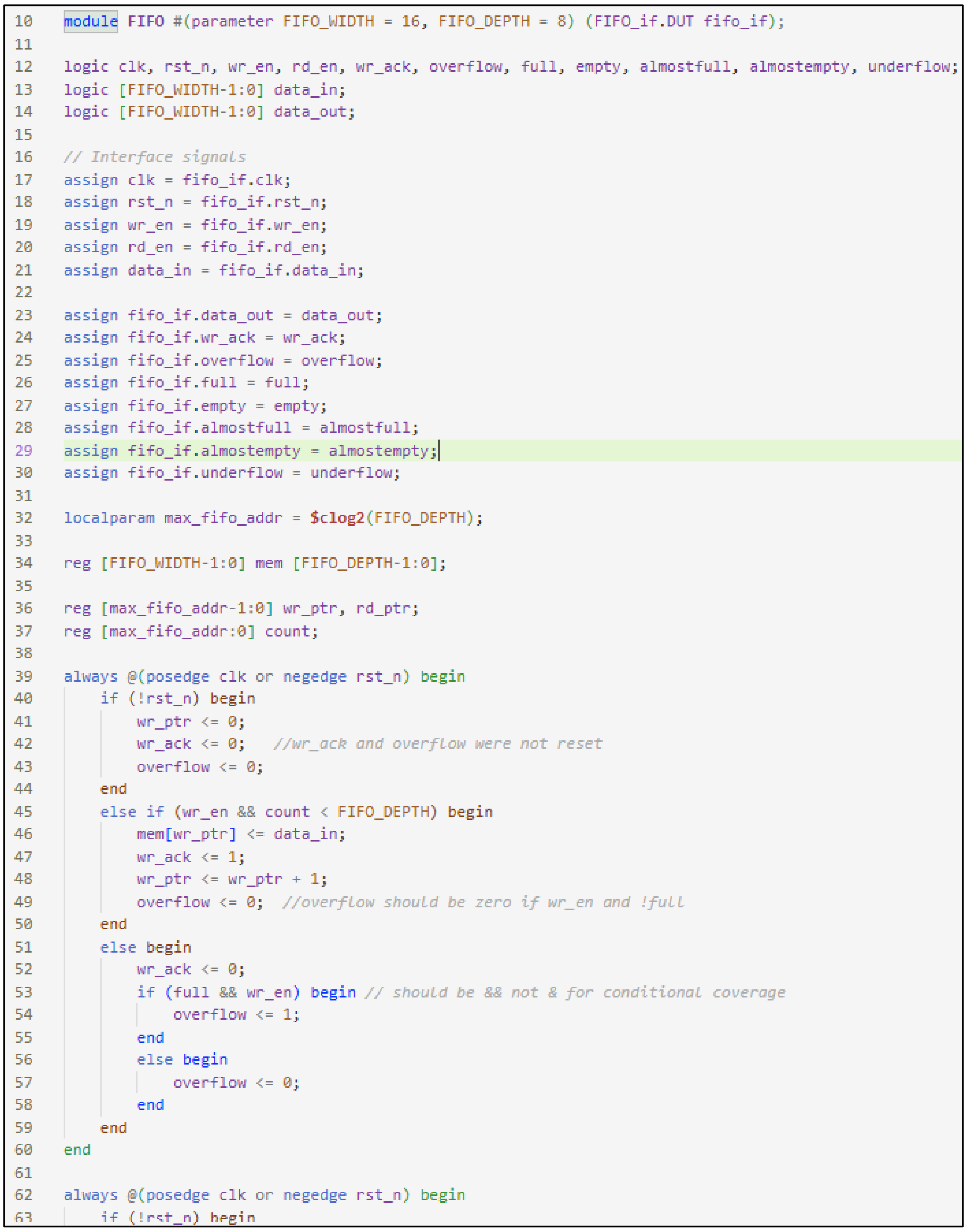


Label	Description	Stimulus Generation	Functional Coverage	Functionality Check
FIFO_1	When the reset_n is asserted, the output count_out value should be low, max_count should be low and zero should be high.	Directed at the start of the simulation, then randomized with constraint that drive the reset to be off(high) most of the simulation time.	-	Immediate assertion to check for the async reset functionality.
FIFO_2	When the rd_en is asserted and wr_en is de-asserted, the output data_out should take the value of the stored current value.	Randomization under constraints on rd_en being on for RD_EN_ON_DISTANCE%.	-	Checked by the reference model. Concurrent assertion to check for the counter decrement.
FIFO_3	When the wr_en is asserted and rd_en is de-asserted, the input data_in should be stored in the fifo, and the output wr_ack should get asserted.	Cyclic randomization on the data_in. Randomization under constraints on wr_en being on for WR_EN_ON_DISTANCE% of the time.	Cover all values of data_in. Cover the crossing of the wr_ack signal with rd_en and wr_en.	Concurrent assertion to check for the wr_ack signal, can also be checked by the reference model. Concurrent assertion to check for the counter increment.
FIFO_4	When the wr_en and rd_en are asserted without empty or full being asserted, the input data_in should be stored in the fifo, and the output wr_ack should get asserted and the current stored value to be read will be output on data_out.	Randomization under constraints on the ce signal to be on(high) 70% of the time.	-	Checked by the reference model. Concurrent assertion to check for the wr_ack signal, can also be checked by the reference model. Concurrent assertion to check for the counter remaining unchanged.
FIFO_5	When the wr_en and rd_en are asserted with empty being asserted, the input data_in should be stored in the fifo, and the output wr_ack should get asserted.	Randomization	Cover the crossing of the wr_ack signal with rd_en and wr_en.	Concurrent assertion to check for the wr_ack signal, can also be checked by the reference model. Concurrent assertion to check for the counter increment.
FIFO_6	When the wr_en and rd_en are asserted with full being asserted, the current stored value to be read will be output on data_out.	Randomization	-	Checked by the reference model. Concurrent assertion to check for the counter decrement.
FIFO_7	When the fifo is full, output flag full must be asserted.	Randomization	Cover the crossing of the full signal with rd_en and wr_en.	Concurrent assertion to check for the full signal, can also be checked by the reference model.
FIFO_8	When the fifo is full and wr_en gets asserted, output flag overflow must be asserted.	Randomization	Cover the crossing of the overflow signal with rd_en and wr_en.	Concurrent assertion to check for the overflow signal, can also be checked by the reference model.
FIFO_9	When the fifo is empty, output flag empty must be asserted.	Randomization	Cover the crossing of the empty signal with rd_en and wr_en.	Concurrent assertion to check for the empty signal, can also be checked by the reference model.
FIFO_10	When the fifo is empty and wr_en gets asserted, output flag underflow must be asserted.	Randomization	Cover the crossing of the underflow signal with rd_en and wr_en.	Concurrent assertion to check for the underflow signal, can also be checked by the reference model.
FIFO_11	When the fifo has only 1 data value stored, output flag almostempty must be asserted.	Randomization	Cover the crossing of the almostempty signal with rd_en and wr_en.	Concurrent assertion to check for the almostempty signal, can also be checked by the reference model.
FIFO_12	When the fifo has 1 remaining spot left to store, output flag almostfull must be asserted.	Randomization	Cover the crossing of the almostfull signal with rd_en and wr_en.	Concurrent assertion to check for the almostfull signal, can also be checked by the reference model.

Figure 1: Sync. FIFO Verification Plan

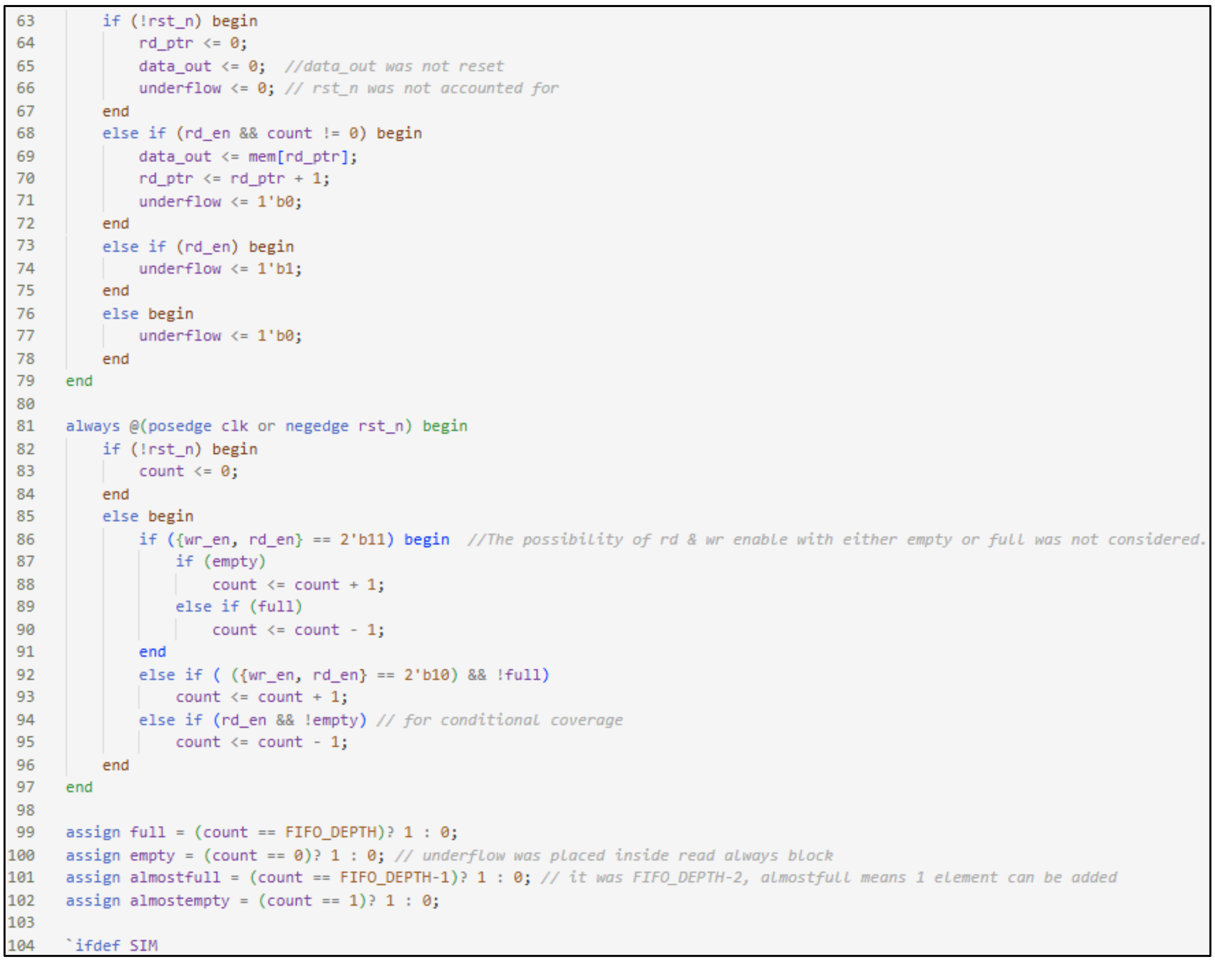
2. Code Snippets

2.1. Design



```
module FIFO #(parameter FIFO_WIDTH = 16, FIFO_DEPTH = 8) (FIFO_if.DUT fifo_if);

logic clk, rst_n, wr_en, rd_en, wr_ack, overflow, full, empty, almostfull, almostempty, underflow;
logic [FIFO_WIDTH-1:0] data_in;
logic [FIFO_WIDTH-1:0] data_out;

// Interface signals
assign clk = fifo_if.clk;
assign rst_n = fifo_if.rst_n;
assign wr_en = fifo_if.wr_en;
assign rd_en = fifo_if.rd_en;
assign data_in = fifo_if.data_in;

assign fifo_if.data_out = data_out;
assign fifo_if.wr_ack = wr_ack;
assign fifo_if.overflow = overflow;
assign fifo_if.full = full;
assign fifo_if.empty = empty;
assign fifo_if.almostfull = almostfull;
assign fifo_if.almostempty = almostempty;
assign fifo_if.underflow = underflow;

localparam max_fifo_addr = $clog2(FIFO_DEPTH);

reg [FIFO_WIDTH-1:0] mem [FIFO_DEPTH-1:0];

reg [max_fifo_addr-1:0] wr_ptr, rd_ptr;
reg [max_fifo_addr:0] count;

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        wr_ptr <= 0;
        wr_ack <= 0;   //wr_ack and overflow were not reset
        overflow <= 0;
    end
    else if (wr_en && count < FIFO_DEPTH) begin
        mem[wr_ptr] <= data_in;
        wr_ack <= 1;
        wr_ptr <= wr_ptr + 1;
        overflow <= 0;  //overflow should be zero if wr_en and !full
    end
    else begin
        wr_ack <= 0;
        if (full && wr_en) begin // should be && not & for conditional coverage
            overflow <= 1;
        end
        else begin
            overflow <= 0;
        end
    end
end

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
```

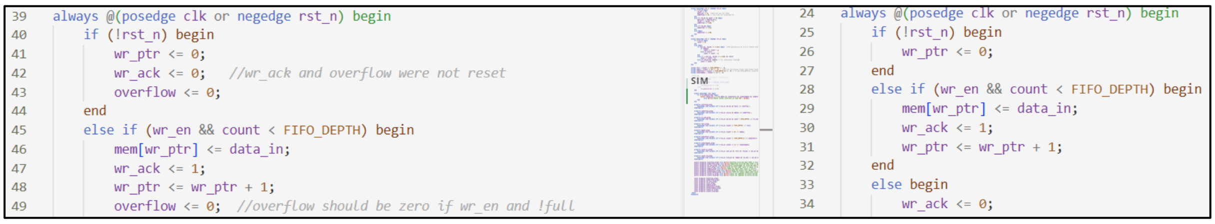
Figure 2: Fixed Design Code



```verilog
    if (!rst_n) begin
        rd_ptr <= 0;
        data_out <= 0;  //data_out was not reset
        underflow <= 0; // rst_n was not accounted for
    end
    else if (rd_en && count != 0) begin
        data_out <= mem[rd_ptr];
        rd_ptr <= rd_ptr + 1;
        underflow <= 1'b0;
    end
    else if (rd_en) begin
        underflow <= 1'b1;
    end
    else begin
        underflow <= 1'b0;
    end
end

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        count <= 0;
    end
    else begin
        if ({wr_en, rd_en} == 2'b11) begin  //The possibility of rd & wr enable with either empty or full was not considered.
            if (empty)
                count <= count + 1;
            else if (full)
                count <= count - 1;
        end
        else if ( ({wr_en, rd_en} == 2'b10) && !full)
            count <= count + 1;
        else if (rd_en && !empty) // for conditional coverage
            count <= count - 1;
    end
end

assign full = (count == FIFO_DEPTH)? 1 : 0;
assign empty = (count == 0)? 1 : 0; // underflow was placed inside read always block
assign almostfull = (count == FIFO_DEPTH-1)? 1 : 0; // it was FIFO_DEPTH-2, almostfull means 1 element can be added
assign almostempty = (count == 1)? 1 : 0;

`ifdef SIM
```

Figure 3: Fixed Design Code (Cont.)

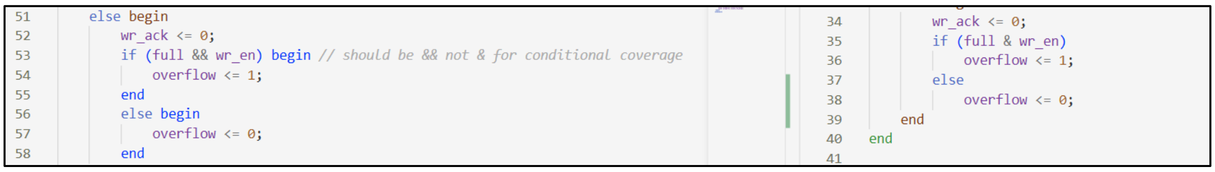
2.1.1. Bugs Found



```verilog
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        wr_ptr <= 0;
        wr_ack <= 0;   //wr_ack and overflow were not reset
        overflow <= 0;
    end
    else if (wr_en && count < FIFO_DEPTH) begin
        mem[wr_ptr] <= data_in;
        wr_ack <= 1;
        wr_ptr <= wr_ptr + 1;
        overflow <= 0;  //overflow should be zero if wr_en and !full
```

```verilog
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        wr_ptr <= 0;
    end
    else if (wr_en && count < FIFO_DEPTH) begin
        mem[wr_ptr] <= data_in;
        wr_ack <= 1;
        wr_ptr <= wr_ptr + 1;
    end
    else begin
        wr_ack <= 0;
```

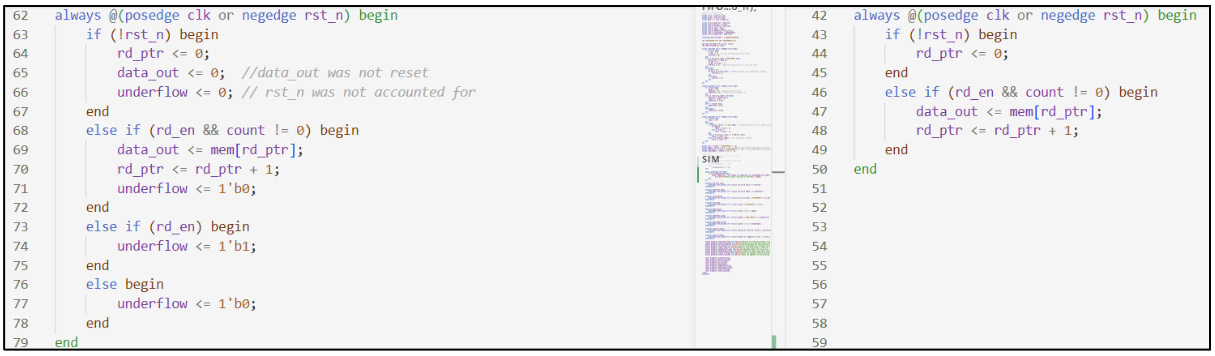
Figure 4: Bugs Found 1



```verilog
    else begin
        wr_ack <= 0;
        if (full && wr_en) begin // should be && not & for conditional coverage
            overflow <= 1;
        end
        else begin
            overflow <= 0;
        end
```

```verilog
        wr_ack <= 0;
        if (full & wr_en)
            overflow <= 1;
        else
            overflow <= 0;
    end
end
```

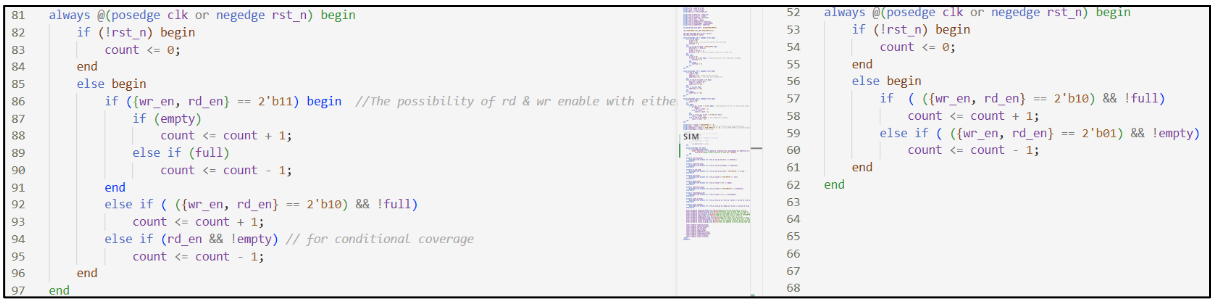
Figure 5: Bugs Found 2



```
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        rd_ptr <= 0;
        data_out <= 0;  //data_out was not reset
        underflow <= 0; // rst_n was not accounted for
    end
    else if (rd_en && count != 0) begin
        data_out <= mem[rd_ptr];
        rd_ptr <= rd_ptr + 1;
        underflow <= 1'b0;
    end
    else if (rd_en) begin
        underflow <= 1'b1;
    end
    else begin
        underflow <= 1'b0;
    end
end
```

```
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        rd_ptr <= 0;
    end
    else if (rd_en && count != 0) begin
        data_out <= mem[rd_ptr];
        rd_ptr <= rd_ptr + 1;
    end
end
```

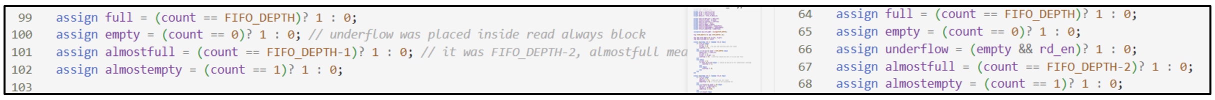
Figure 6: Bugs Found 3



```
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        count <= 0;
    end
    else begin
        if ({wr_en, rd_en} == 2'b11) begin  //The possibility of rd & wr enable with eithe
            if (empty)
                count <= count + 1;
            else if (full)
                count <= count - 1;
        end
        else if ( ({wr_en, rd_en} == 2'b10) && !full)
            count <= count + 1;
        else if (rd_en && !empty) // for conditional coverage
            count <= count - 1;
    end
end
```

```
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        count <= 0;
    end
    else begin
        if  ( ({wr_en, rd_en} == 2'b10) && !full)
            count <= count + 1;
        else if ( ({wr_en, rd_en} == 2'b01) && !empty)
            count <= count - 1;
    end
end
```

Figure 7: Bugs Found 4

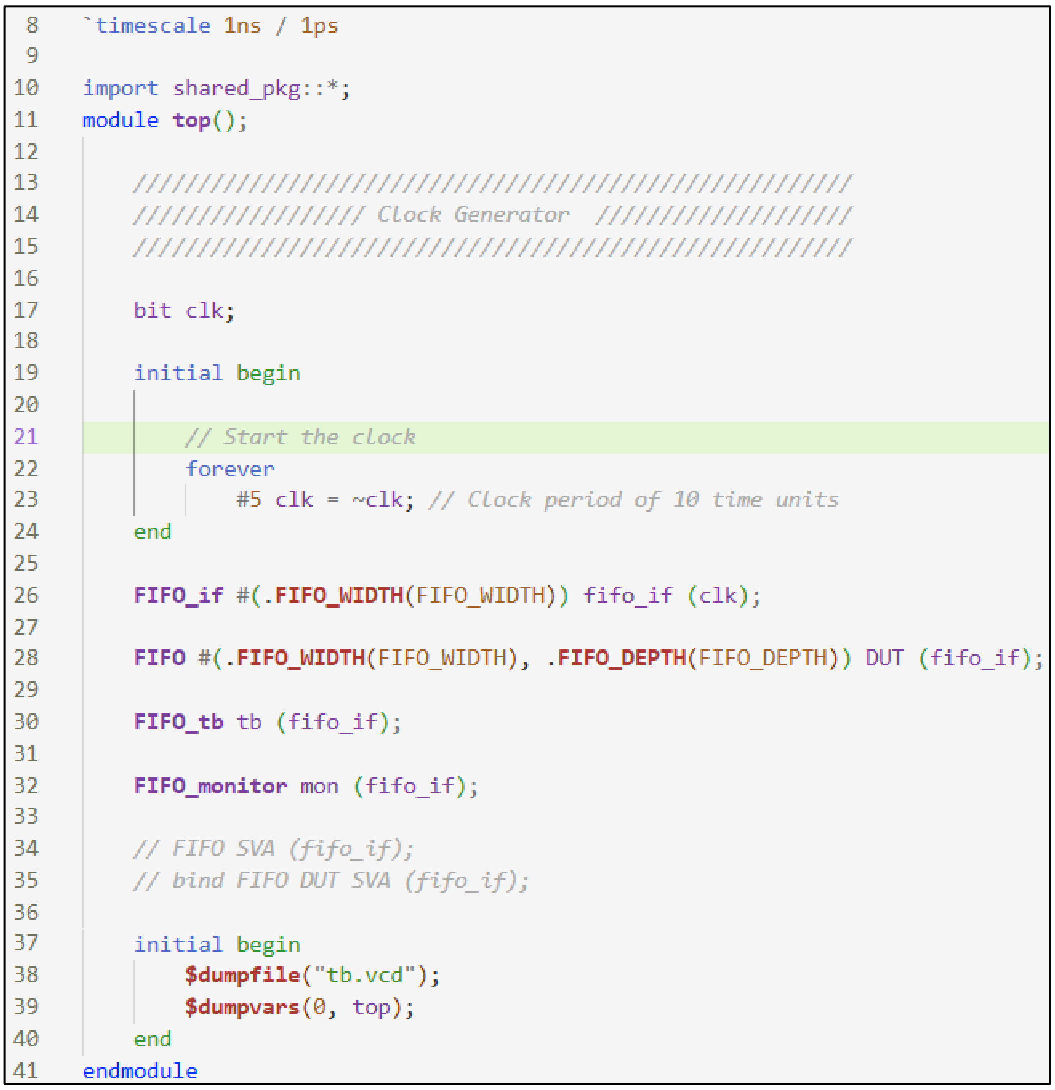


```
assign full = (count == FIFO_DEPTH)? 1 : 0;
assign empty = (count == 0)? 1 : 0; // underflow was placed inside read always block
assign almostfull = (count == FIFO_DEPTH-1)? 1 : 0; // it was FIFO_DEPTH-2, almostfull mea
assign almostempty = (count == 1)? 1 : 0;
```

```
assign full = (count == FIFO_DEPTH)? 1 : 0;
assign empty = (count == 0)? 1 : 0;
assign underflow = (empty && rd_en)? 1 : 0;
assign almostfull = (count == FIFO_DEPTH-2)? 1 : 0;
assign almostempty = (count == 1)? 1 : 0;
```

Figure 8: Bugs Found 5

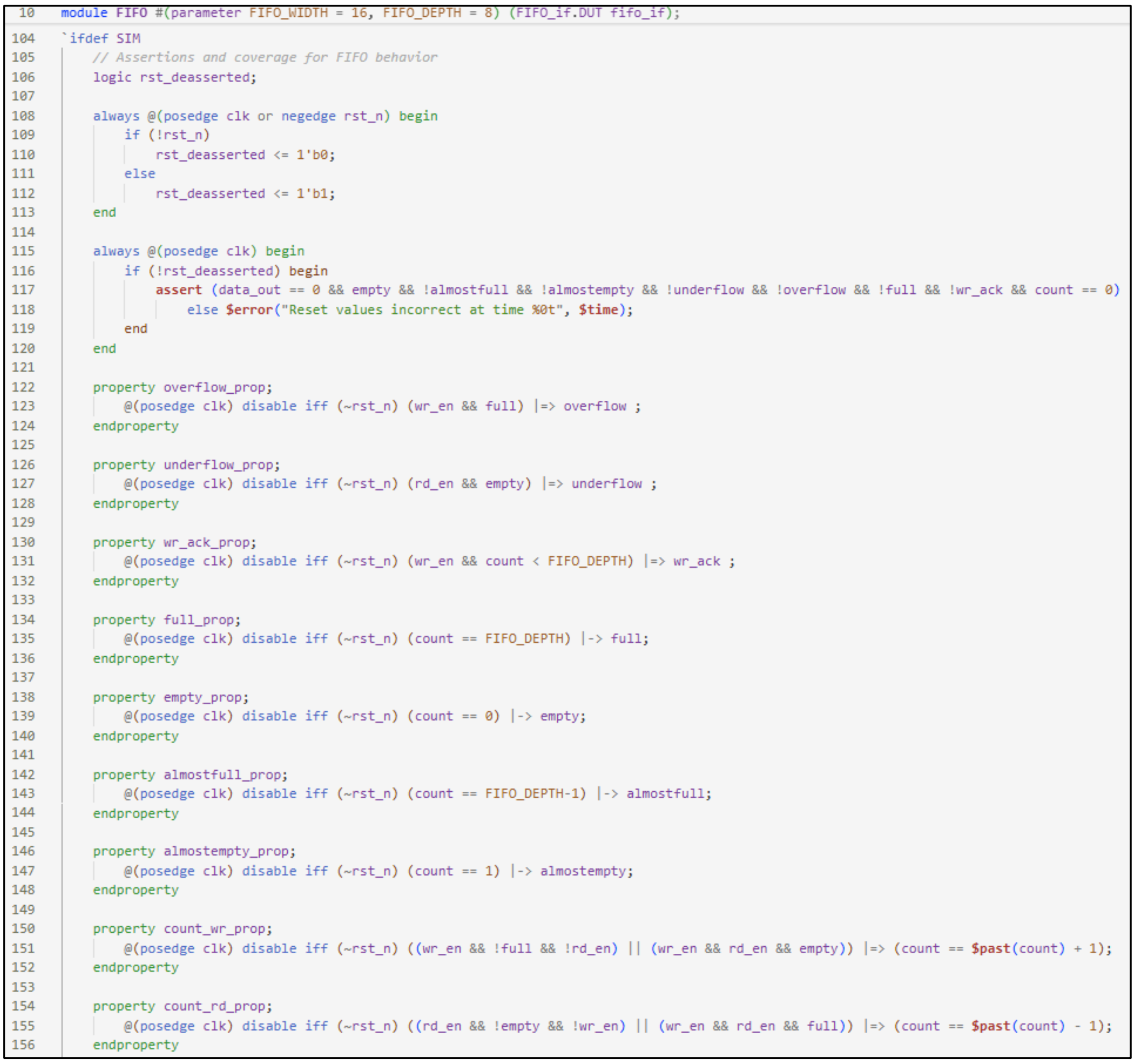
2.2. Top Module



```
`timescale 1ns / 1ps

import shared_pkg::*;
module top();

    ////////////////////////////////////////////////////////
    ////////////////// Clock Generator  ////////////////////
    ////////////////////////////////////////////////////////

    bit clk;

    initial begin

        // Start the clock
        forever
            #5 clk = ~clk; // Clock period of 10 time units
    end

    FIFO_if #(.FIFO_WIDTH(FIFO_WIDTH)) fifo_if (clk);

    FIFO #(.FIFO_WIDTH(FIFO_WIDTH), .FIFO_DEPTH(FIFO_DEPTH)) DUT (fifo_if);

    FIFO_tb tb (fifo_if);

    FIFO_monitor mon (fifo_if);

    // FIFO SVA (fifo_if);
    // bind FIFO DUT SVA (fifo_if);

    initial begin
        $dumpfile("tb.vcd");
        $dumpvars(0, top);
    end
endmodule
```

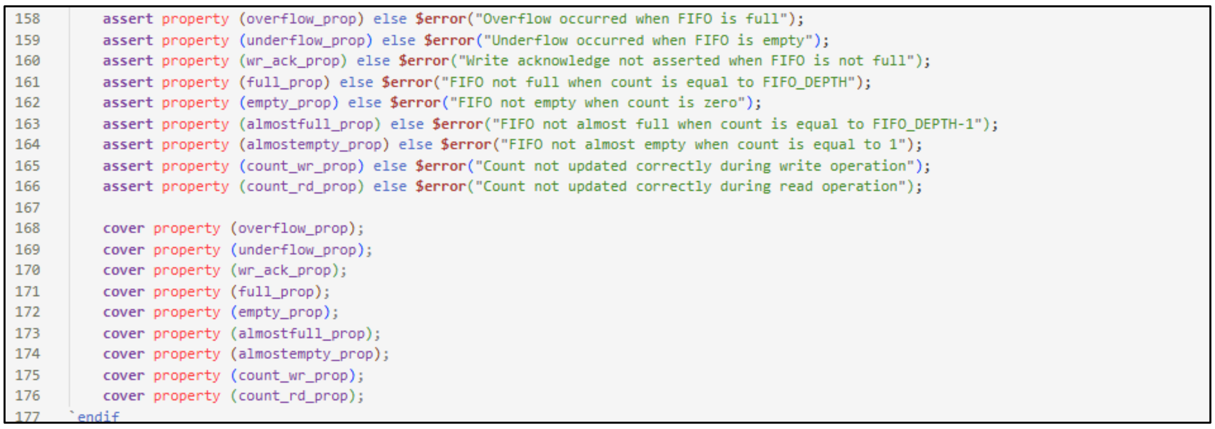
Figure 9: Top Module Code

2.3. SVAs



```systemverilog
module FIFO #(parameter FIFO_WIDTH = 16, FIFO_DEPTH = 8) (FIFO_if.DUT fifo_if);

`ifdef SIM
    // Assertions and coverage for FIFO behavior
    logic rst_deasserted;

    always @(posedge clk or negedge rst_n) begin
        if (!rst_n)
            rst_deasserted <= 1'b0;
        else
            rst_deasserted <= 1'b1;
    end

    always @(posedge clk) begin
        if (!rst_deasserted) begin
            assert (data_out == 0 && empty && !almostfull && !almostempty && !underflow && !overflow && !full && !wr_ack && count == 0)
                else $error("Reset values incorrect at time %0t", $time);
        end
    end

    property overflow_prop;
        @(posedge clk) disable iff (~rst_n) (wr_en && full) |=> overflow ;
    endproperty

    property underflow_prop;
        @(posedge clk) disable iff (~rst_n) (rd_en && empty) |=> underflow ;
    endproperty

    property wr_ack_prop;
        @(posedge clk) disable iff (~rst_n) (wr_en && count < FIFO_DEPTH) |=> wr_ack ;
    endproperty

    property full_prop;
        @(posedge clk) disable iff (~rst_n) (count == FIFO_DEPTH) |-> full;
    endproperty

    property empty_prop;
        @(posedge clk) disable iff (~rst_n) (count == 0) |-> empty;
    endproperty

    property almostfull_prop;
        @(posedge clk) disable iff (~rst_n) (count == FIFO_DEPTH-1) |-> almostfull;
    endproperty

    property almostempty_prop;
        @(posedge clk) disable iff (~rst_n) (count == 1) |-> almostempty;
    endproperty

    property count_wr_prop;
        @(posedge clk) disable iff (~rst_n) ((wr_en && !full && !rd_en) || (wr_en && rd_en && empty)) |=> (count == $past(count) + 1);
    endproperty

    property count_rd_prop;
        @(posedge clk) disable iff (~rst_n) ((rd_en && !empty && !wr_en) || (wr_en && rd_en && full)) |=> (count == $past(count) - 1);
    endproperty
```

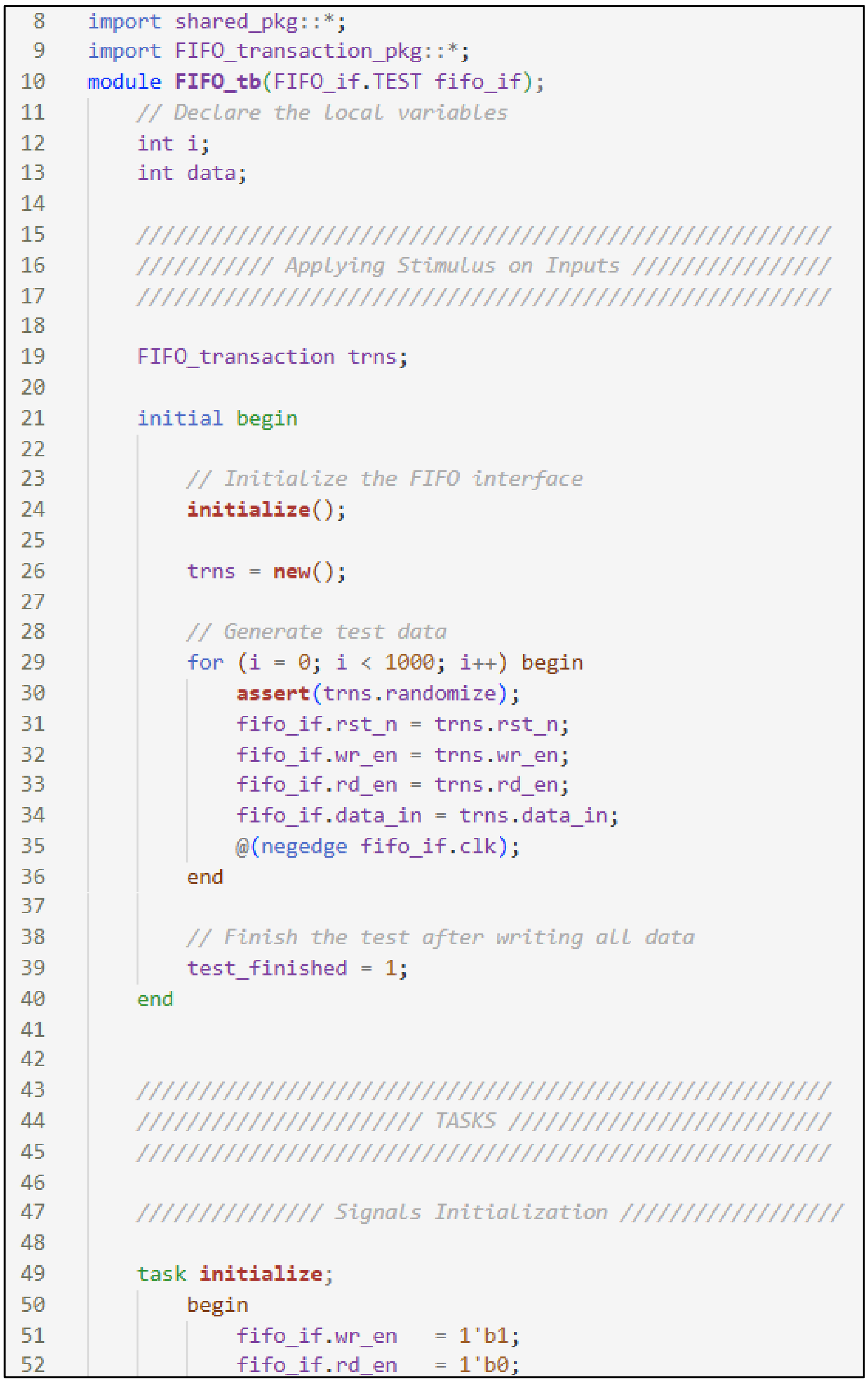
Figure 10: Properties Code



```systemverilog
    assert property (overflow_prop) else $error("Overflow occurred when FIFO is full");
    assert property (underflow_prop) else $error("Underflow occurred when FIFO is empty");
    assert property (wr_ack_prop) else $error("Write acknowledge not asserted when FIFO is not full");
    assert property (full_prop) else $error("FIFO not full when count is equal to FIFO_DEPTH");
    assert property (empty_prop) else $error("FIFO not empty when count is zero");
    assert property (almostfull_prop) else $error("FIFO not almost full when count is equal to FIFO_DEPTH-1");
    assert property (almostempty_prop) else $error("FIFO not almost empty when count is equal to 1");
    assert property (count_wr_prop) else $error("Count not updated correctly during write operation");
    assert property (count_rd_prop) else $error("Count not updated correctly during read operation");

    cover property (overflow_prop);
    cover property (underflow_prop);
    cover property (wr_ack_prop);
    cover property (full_prop);
    cover property (empty_prop);
    cover property (almostfull_prop);
    cover property (almostempty_prop);
    cover property (count_wr_prop);
    cover property (count_rd_prop);
`endif
```

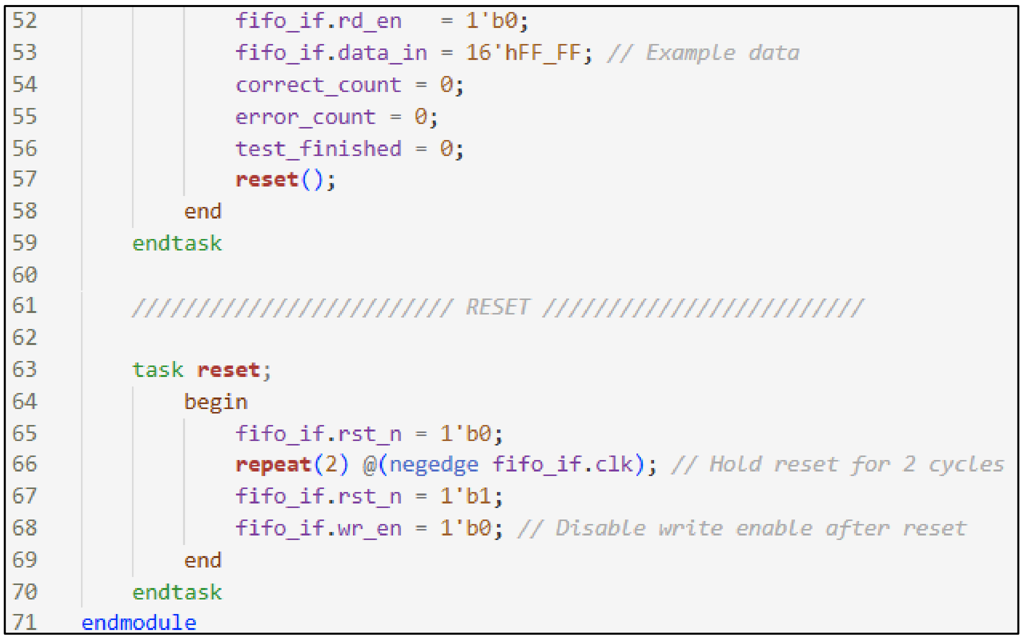
Figure 11: Assertions and Covers Code

2.4. Test Bench Module



```systemverilog
import shared_pkg::*;
import FIFO_transaction_pkg::*;
module FIFO_tb(FIFO_if.TEST fifo_if);
    // Declare the local variables
    int i;
    int data;

    ////////////////////////////////////////////////////////
    /////////// Applying Stimulus on Inputs ////////////////
    ////////////////////////////////////////////////////////

    FIFO_transaction trns;

    initial begin

        // Initialize the FIFO interface
        initialize();

        trns = new();

        // Generate test data
        for (i = 0; i < 1000; i++) begin
            assert(trns.randomize);
            fifo_if.rst_n = trns.rst_n;
            fifo_if.wr_en = trns.wr_en;
            fifo_if.rd_en = trns.rd_en;
            fifo_if.data_in = trns.data_in;
            @(negedge fifo_if.clk);
        end

        // Finish the test after writing all data
        test_finished = 1;
    end


    ////////////////////////////////////////////////////////
    /////////////////////// TASKS //////////////////////////
    ////////////////////////////////////////////////////////

    /////////////// Signals Initialization //////////////////

    task initialize;
        begin
            fifo_if.wr_en    = 1'b1;
            fifo_if.rd_en    = 1'b0;
```

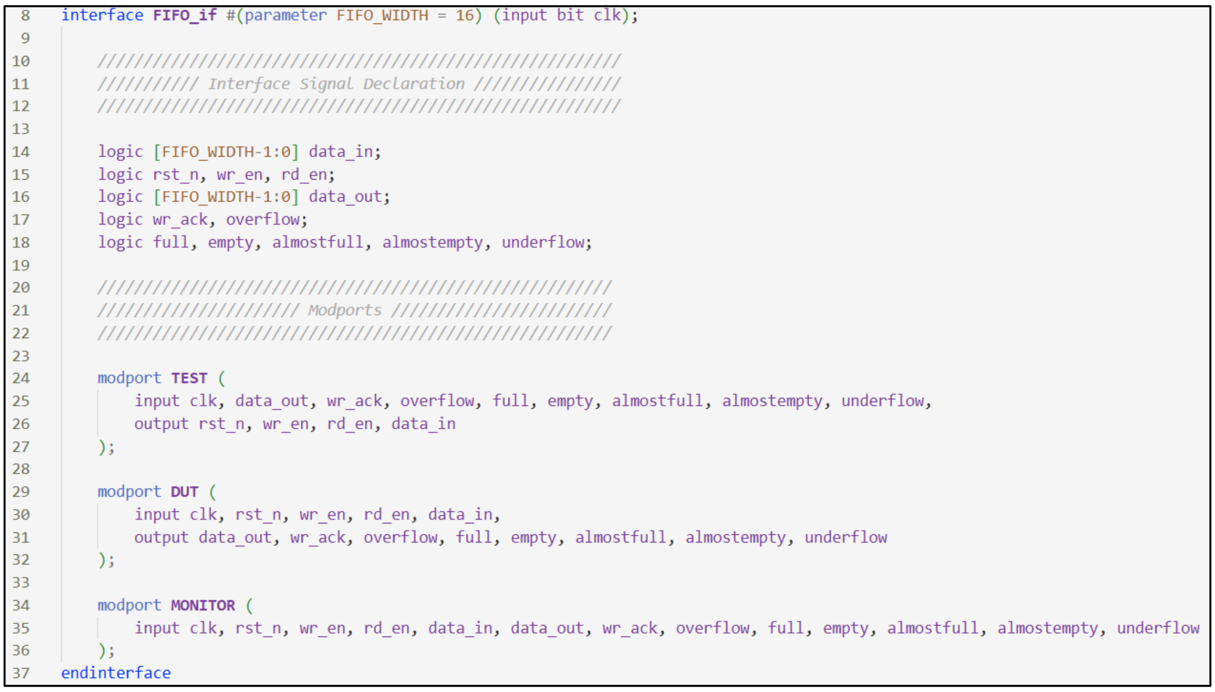
Figure 12: Test Bench Module Code



```
            fifo_if.rd_en   = 1'b0;
            fifo_if.data_in = 16'hFF_FF; // Example data
            correct_count = 0;
            error_count = 0;
            test_finished = 0;
            reset();
        end
    endtask

    //////////////////////// RESET ////////////////////////

    task reset;
        begin
            fifo_if.rst_n = 1'b0;
            repeat(2) @(negedge fifo_if.clk); // Hold reset for 2 cycles
            fifo_if.rst_n = 1'b1;
            fifo_if.wr_en = 1'b0; // Disable write enable after reset
        end
    endtask
endmodule
```

Figure 13: Test Bench Module Code (Cont.)

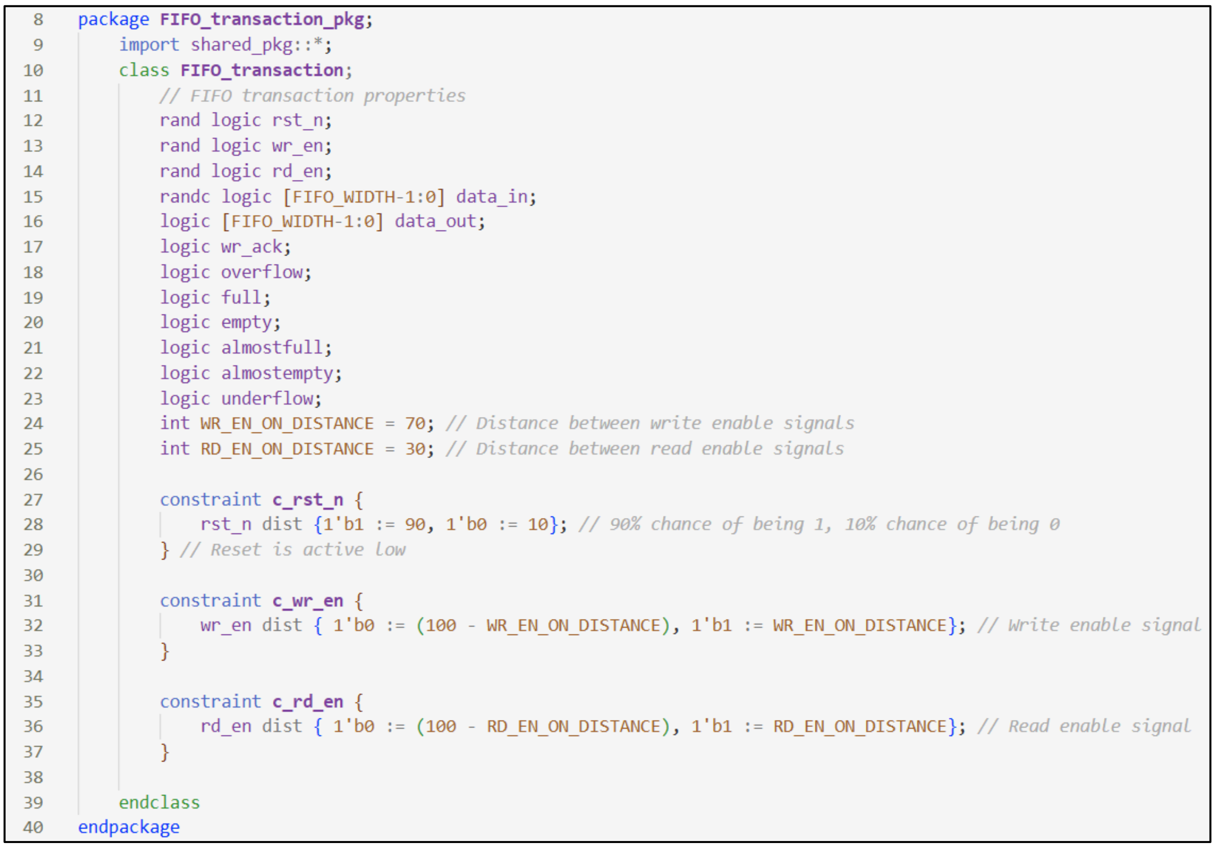
2.5. Interface



```
interface FIFO_if #(parameter FIFO_WIDTH = 16) (input bit clk);

    ////////////////////////////////////////////////////////
    ////////// Interface Signal Declaration ///////////////
    ////////////////////////////////////////////////////////

    logic [FIFO_WIDTH-1:0] data_in;
    logic rst_n, wr_en, rd_en;
    logic [FIFO_WIDTH-1:0] data_out;
    logic wr_ack, overflow;
    logic full, empty, almostfull, almostempty, underflow;

    ///////////////////////////////////////////////////////
    ///////////////////// Modports //////////////////////
    ///////////////////////////////////////////////////////

    modport TEST (
        input clk, data_out, wr_ack, overflow, full, empty, almostfull, almostempty, underflow,
        output rst_n, wr_en, rd_en, data_in
    );

    modport DUT (
        input clk, rst_n, wr_en, rd_en, data_in,
        output data_out, wr_ack, overflow, full, empty, almostfull, almostempty, underflow
    );

    modport MONITOR (
        input clk, rst_n, wr_en, rd_en, data_in, data_out, wr_ack, overflow, full, empty, almostfull, almostempty, underflow
    );
endinterface
```

Figure 14: Interface Code

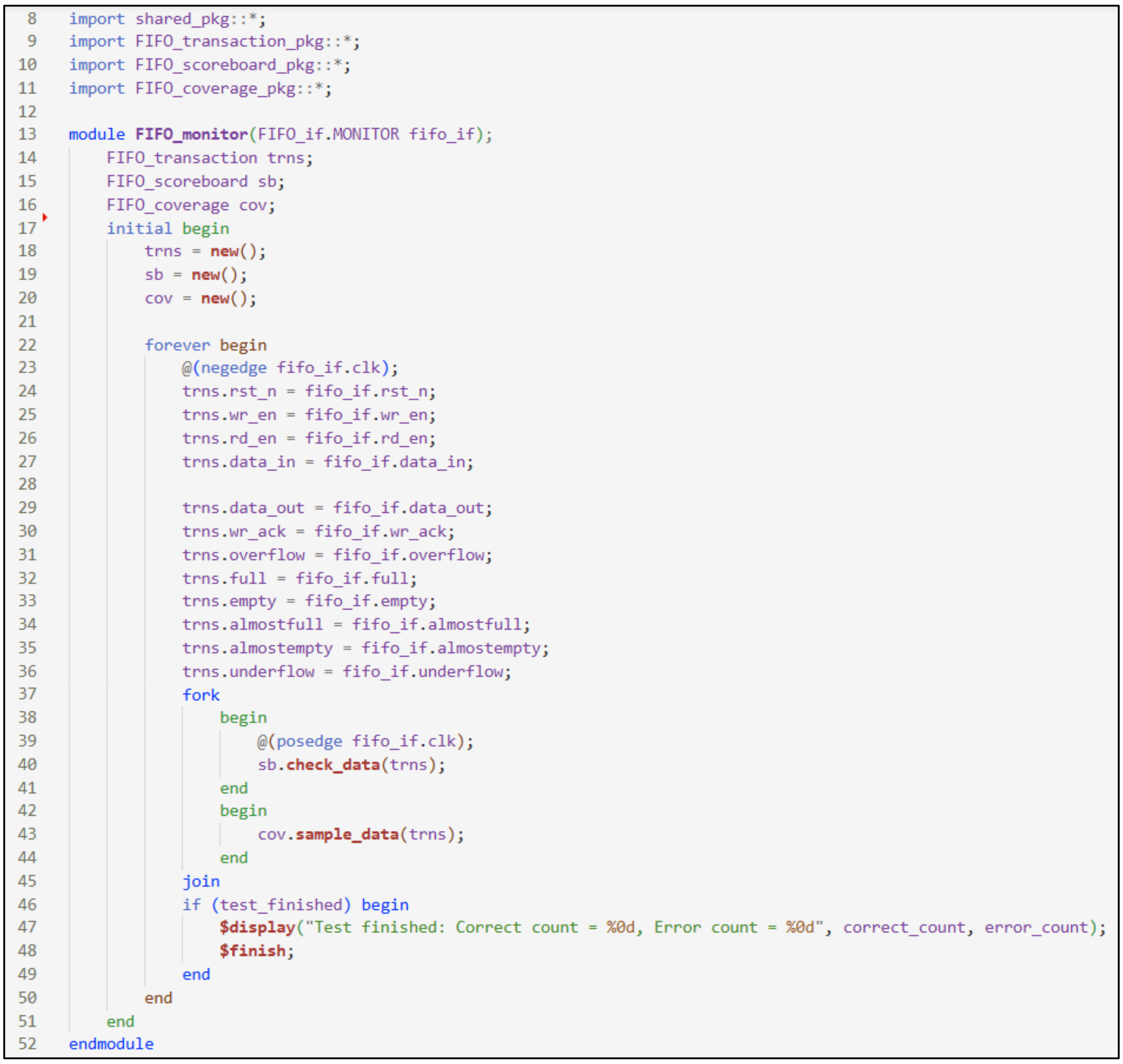
2.6. Transaction Package



```systemverilog
package FIFO_transaction_pkg;
    import shared_pkg::*;
    class FIFO_transaction;
        // FIFO transaction properties
        rand logic rst_n;
        rand logic wr_en;
        rand logic rd_en;
        randc logic [FIFO_WIDTH-1:0] data_in;
        logic [FIFO_WIDTH-1:0] data_out;
        logic wr_ack;
        logic overflow;
        logic full;
        logic empty;
        logic almostfull;
        logic almostempty;
        logic underflow;
        int WR_EN_ON_DISTANCE = 70; // Distance between write enable signals
        int RD_EN_ON_DISTANCE = 30; // Distance between read enable signals

        constraint c_rst_n {
            rst_n dist {1'b1 := 90, 1'b0 := 10}; // 90% chance of being 1, 10% chance of being 0
        } // Reset is active low

        constraint c_wr_en {
            wr_en dist { 1'b0 := (100 - WR_EN_ON_DISTANCE), 1'b1 := WR_EN_ON_DISTANCE}; // Write enable signal
        }

        constraint c_rd_en {
            rd_en dist { 1'b0 := (100 - RD_EN_ON_DISTANCE), 1'b1 := RD_EN_ON_DISTANCE}; // Read enable signal
        }

    endclass
endpackage
```

Figure 15: Transaction Package Code

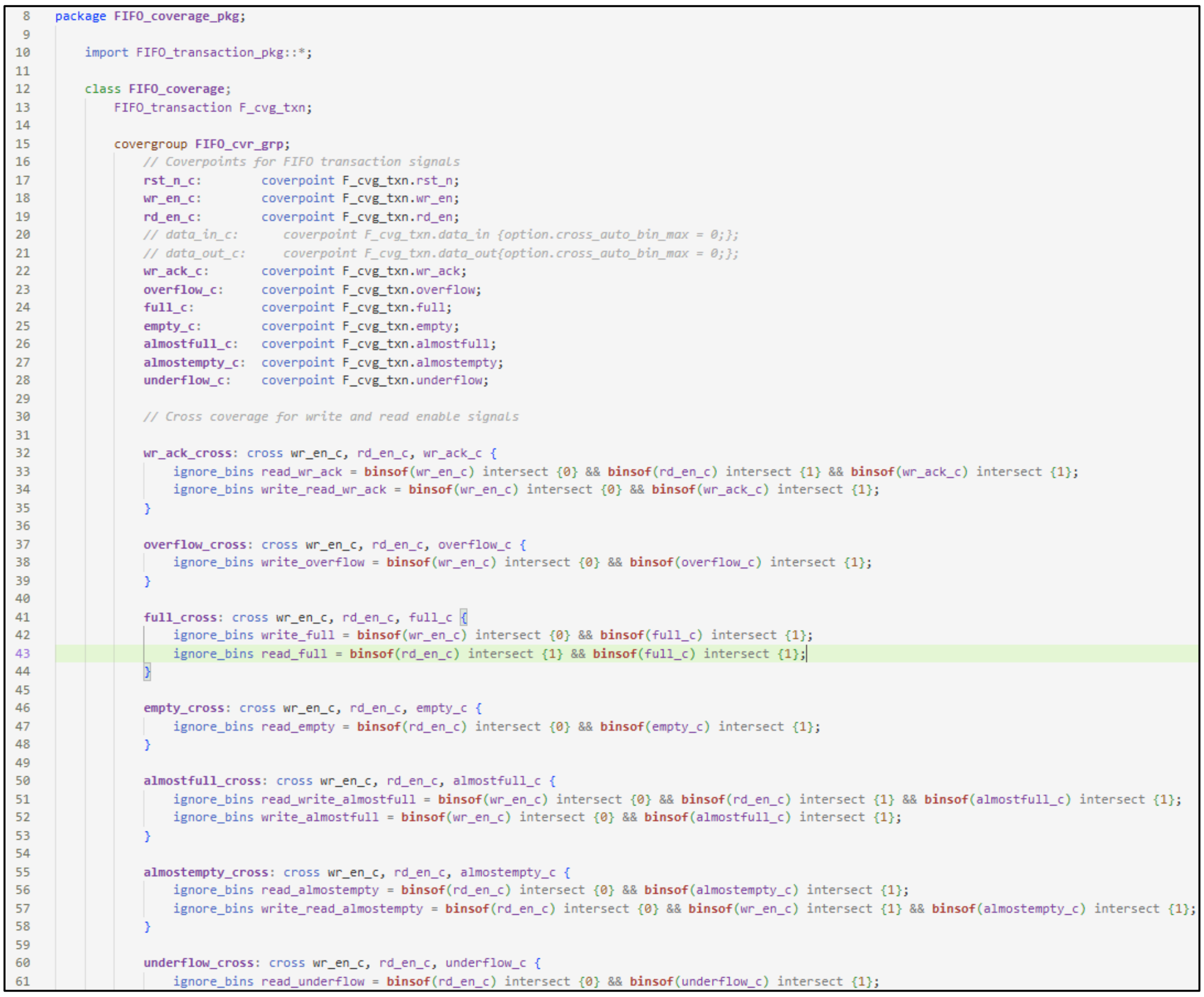
2.7. Monitor Module



```systemverilog
import shared_pkg::*;
import FIFO_transaction_pkg::*;
import FIFO_scoreboard_pkg::*;
import FIFO_coverage_pkg::*;

module FIFO_monitor(FIFO_if.MONITOR fifo_if);
    FIFO_transaction trns;
    FIFO_scoreboard sb;
    FIFO_coverage cov;
    initial begin
        trns = new();
        sb = new();
        cov = new();

        forever begin
            @(negedge fifo_if.clk);
            trns.rst_n = fifo_if.rst_n;
            trns.wr_en = fifo_if.wr_en;
            trns.rd_en = fifo_if.rd_en;
            trns.data_in = fifo_if.data_in;

            trns.data_out = fifo_if.data_out;
            trns.wr_ack = fifo_if.wr_ack;
            trns.overflow = fifo_if.overflow;
            trns.full = fifo_if.full;
            trns.empty = fifo_if.empty;
            trns.almostfull = fifo_if.almostfull;
            trns.almostempty = fifo_if.almostempty;
            trns.underflow = fifo_if.underflow;
            fork
                begin
                    @(posedge fifo_if.clk);
                    sb.check_data(trns);
                end
                begin
                    cov.sample_data(trns);
                end
            join
            if (test_finished) begin
                $display("Test finished: Correct count = %0d, Error count = %0d", correct_count, error_count);
                $finish;
            end
        end
    end
endmodule
```

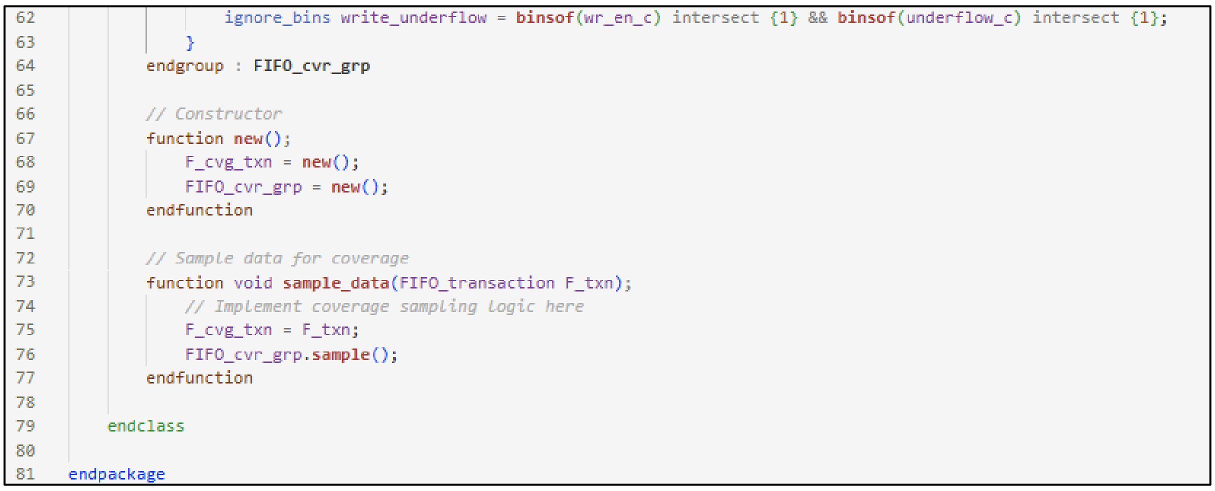
Figure 16: Monitor Module Code

2.8. Coverage Package



```
package FIFO_coverage_pkg;

    import FIFO_transaction_pkg::*;

    class FIFO_coverage;
        FIFO_transaction F_cvg_txn;

        covergroup FIFO_cvr_grp;
            // Coverpoints for FIFO transaction signals
            rst_n_c:        coverpoint F_cvg_txn.rst_n;
            wr_en_c:        coverpoint F_cvg_txn.wr_en;
            rd_en_c:        coverpoint F_cvg_txn.rd_en;
            // data_in_c:      coverpoint F_cvg_txn.data_in {option.cross_auto_bin_max = 0;};
            // data_out_c:     coverpoint F_cvg_txn.data_out{option.cross_auto_bin_max = 0;};
            wr_ack_c:       coverpoint F_cvg_txn.wr_ack;
            overflow_c:     coverpoint F_cvg_txn.overflow;
            full_c:         coverpoint F_cvg_txn.full;
            empty_c:        coverpoint F_cvg_txn.empty;
            almostfull_c:   coverpoint F_cvg_txn.almostfull;
            almostempty_c:  coverpoint F_cvg_txn.almostempty;
            underflow_c:    coverpoint F_cvg_txn.underflow;

            // Cross coverage for write and read enable signals

            wr_ack_cross: cross wr_en_c, rd_en_c, wr_ack_c {
                ignore_bins read_wr_ack = binsof(wr_en_c) intersect {0} && binsof(rd_en_c) intersect {1} && binsof(wr_ack_c) intersect {1};
                ignore_bins write_read_wr_ack = binsof(wr_en_c) intersect {0} && binsof(wr_ack_c) intersect {1};
            }

            overflow_cross: cross wr_en_c, rd_en_c, overflow_c {
                ignore_bins write_overflow = binsof(wr_en_c) intersect {0} && binsof(overflow_c) intersect {1};
            }

            full_cross: cross wr_en_c, rd_en_c, full_c {
                ignore_bins write_full = binsof(wr_en_c) intersect {0} && binsof(full_c) intersect {1};
                ignore_bins read_full = binsof(rd_en_c) intersect {1} && binsof(full_c) intersect {1};
            }

            empty_cross: cross wr_en_c, rd_en_c, empty_c {
                ignore_bins read_empty = binsof(rd_en_c) intersect {0} && binsof(empty_c) intersect {1};
            }

            almostfull_cross: cross wr_en_c, rd_en_c, almostfull_c {
                ignore_bins read_write_almostfull = binsof(wr_en_c) intersect {0} && binsof(rd_en_c) intersect {1} && binsof(almostfull_c) intersect {1};
                ignore_bins write_almostfull = binsof(wr_en_c) intersect {0} && binsof(almostfull_c) intersect {1};
            }

            almostempty_cross: cross wr_en_c, rd_en_c, almostempty_c {
                ignore_bins read_almostempty = binsof(rd_en_c) intersect {0} && binsof(almostempty_c) intersect {1};
                ignore_bins write_read_almostempty = binsof(rd_en_c) intersect {0} && binsof(wr_en_c) intersect {1} && binsof(almostempty_c) intersect {1};
            }

            underflow_cross: cross wr_en_c, rd_en_c, underflow_c {
                ignore_bins read_underflow = binsof(rd_en_c) intersect {0} && binsof(underflow_c) intersect {1};
```

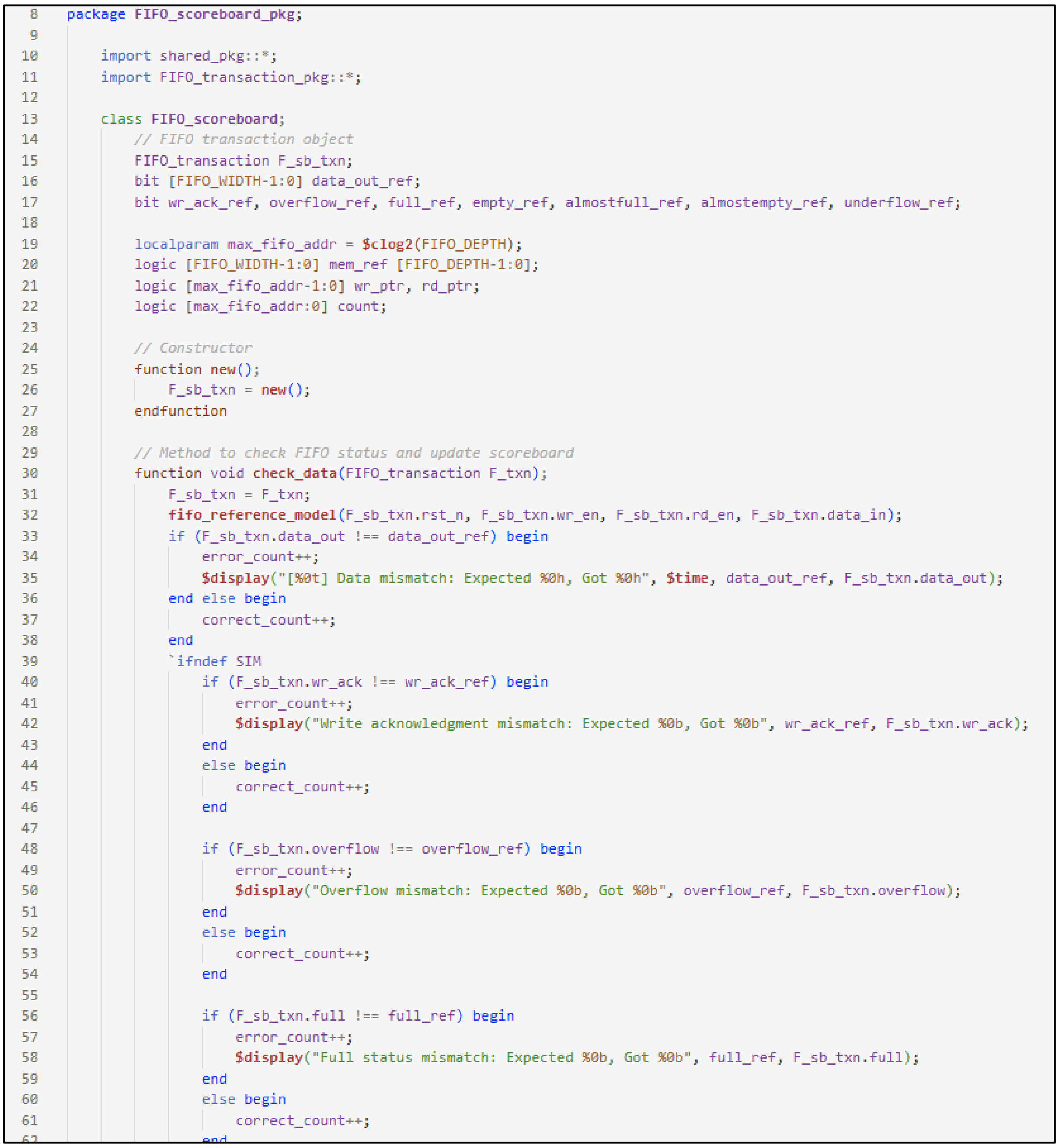
Figure 17: Coverage Package Code



```
                ignore_bins write_underflow = binsof(wr_en_c) intersect {1} && binsof(underflow_c) intersect {1};
            }
        endgroup : FIFO_cvr_grp

        // Constructor
        function new();
            F_cvg_txn = new();
            FIFO_cvr_grp = new();
        endfunction

        // Sample data for coverage
        function void sample_data(FIFO_transaction F_txn);
            // Implement coverage sampling logic here
            F_cvg_txn = F_txn;
            FIFO_cvr_grp.sample();
        endfunction

    endclass

endpackage
```

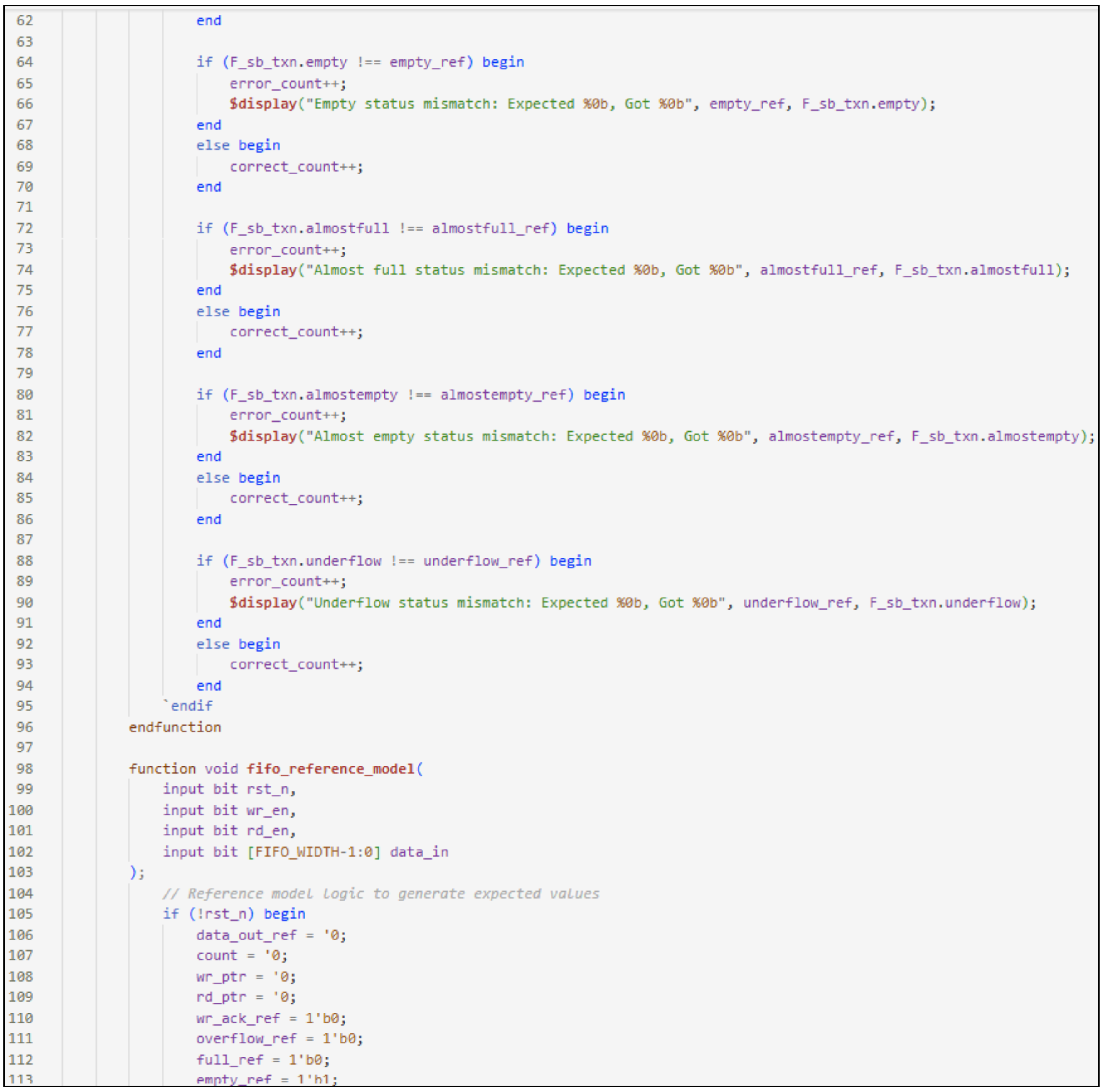
Figure 18: Coverage Package Code (Cont.)

2.9. Scoreboard Package



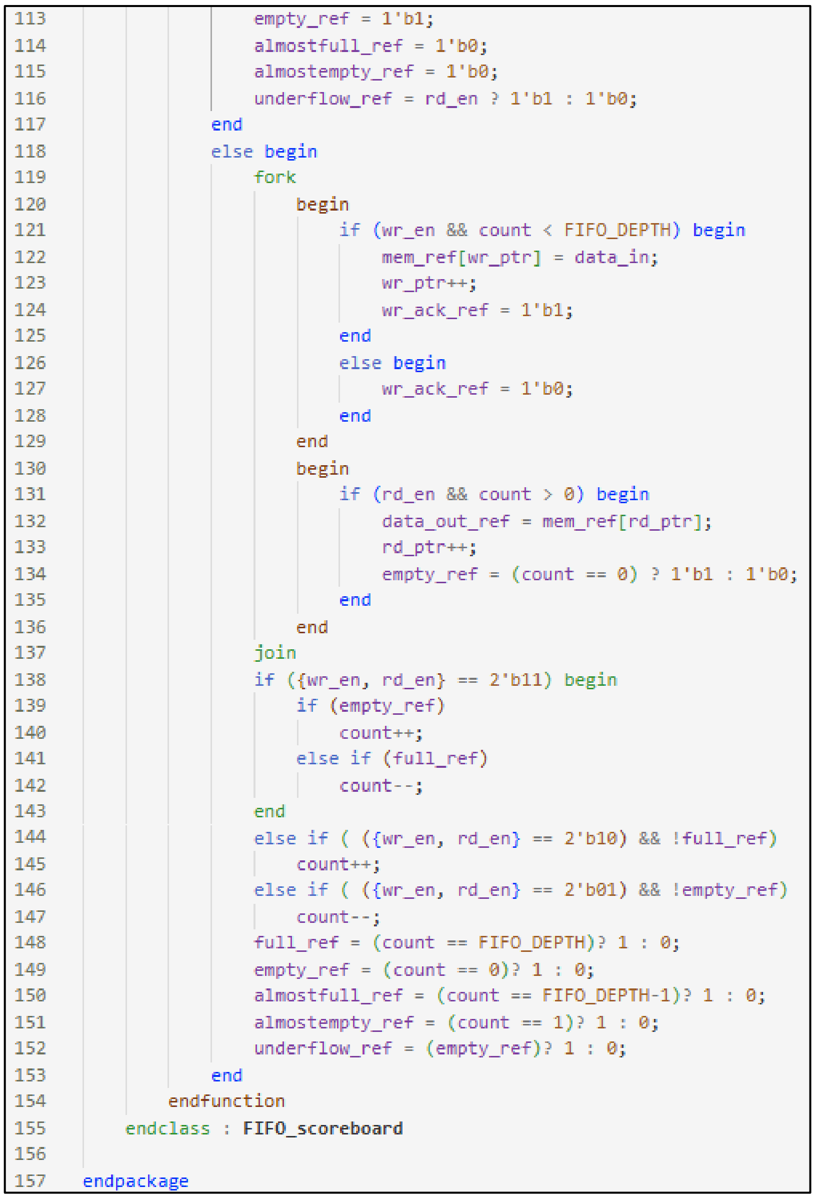
```
package FIFO_scoreboard_pkg;

    import shared_pkg::*;
    import FIFO_transaction_pkg::*;

    class FIFO_scoreboard;
        // FIFO transaction object
        FIFO_transaction F_sb_txn;
        bit [FIFO_WIDTH-1:0] data_out_ref;
        bit wr_ack_ref, overflow_ref, full_ref, empty_ref, almostfull_ref, almostempty_ref, underflow_ref;

        localparam max_fifo_addr = $clog2(FIFO_DEPTH);
        logic [FIFO_WIDTH-1:0] mem_ref [FIFO_DEPTH-1:0];
        logic [max_fifo_addr-1:0] wr_ptr, rd_ptr;
        logic [max_fifo_addr:0] count;

        // Constructor
        function new();
            F_sb_txn = new();
        endfunction

        // Method to check FIFO status and update scoreboard
        function void check_data(FIFO_transaction F_txn);
            F_sb_txn = F_txn;
            fifo_reference_model(F_sb_txn.rst_n, F_sb_txn.wr_en, F_sb_txn.rd_en, F_sb_txn.data_in);
            if (F_sb_txn.data_out !== data_out_ref) begin
                error_count++;
                $display("[%0t] Data mismatch: Expected %0h, Got %0h", $time, data_out_ref, F_sb_txn.data_out);
            end else begin
                correct_count++;
            end
            `ifndef SIM
                if (F_sb_txn.wr_ack !== wr_ack_ref) begin
                    error_count++;
                    $display("Write acknowledgment mismatch: Expected %0b, Got %0b", wr_ack_ref, F_sb_txn.wr_ack);
                end
                else begin
                    correct_count++;
                end

                if (F_sb_txn.overflow !== overflow_ref) begin
                    error_count++;
                    $display("Overflow mismatch: Expected %0b, Got %0b", overflow_ref, F_sb_txn.overflow);
                end
                else begin
                    correct_count++;
                end

                if (F_sb_txn.full !== full_ref) begin
                    error_count++;
                    $display("Full status mismatch: Expected %0b, Got %0b", full_ref, F_sb_txn.full);
                end
                else begin
                    correct_count++;
                end
```

Figure 19: Scoreboard Package Code 1



```
                end

                if (F_sb_txn.empty !== empty_ref) begin
                    error_count++;
                    $display("Empty status mismatch: Expected %0b, Got %0b", empty_ref, F_sb_txn.empty);
                end
                else begin
                    correct_count++;
                end

                if (F_sb_txn.almostfull !== almostfull_ref) begin
                    error_count++;
                    $display("Almost full status mismatch: Expected %0b, Got %0b", almostfull_ref, F_sb_txn.almostfull);
                end
                else begin
                    correct_count++;
                end

                if (F_sb_txn.almostempty !== almostempty_ref) begin
                    error_count++;
                    $display("Almost empty status mismatch: Expected %0b, Got %0b", almostempty_ref, F_sb_txn.almostempty);
                end
                else begin
                    correct_count++;
                end

                if (F_sb_txn.underflow !== underflow_ref) begin
                    error_count++;
                    $display("Underflow status mismatch: Expected %0b, Got %0b", underflow_ref, F_sb_txn.underflow);
                end
                else begin
                    correct_count++;
                end
            `endif
        endfunction

        function void fifo_reference_model(
            input bit rst_n,
            input bit wr_en,
            input bit rd_en,
            input bit [FIFO_WIDTH-1:0] data_in
        );
            // Reference model logic to generate expected values
            if (!rst_n) begin
                data_out_ref = '0;
                count = '0;
                wr_ptr = '0;
                rd_ptr = '0;
                wr_ack_ref = 1'b0;
                overflow_ref = 1'b0;
                full_ref = 1'b0;
                empty_ref = 1'b1;
```

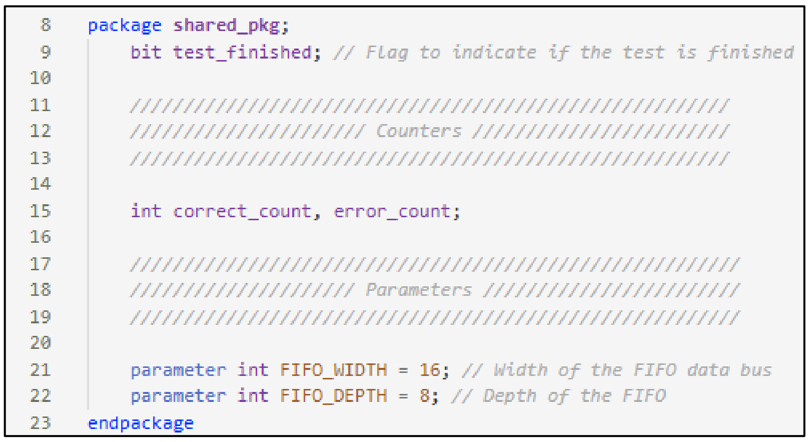
Figure 20: Scoreboard Package Code 2



```
                empty_ref = 1'b1;
                almostfull_ref = 1'b0;
                almostempty_ref = 1'b0;
                underflow_ref = rd_en ? 1'b1 : 1'b0;
            end
            else begin
                fork
                    begin
                        if (wr_en && count < FIFO_DEPTH) begin
                            mem_ref[wr_ptr] = data_in;
                            wr_ptr++;
                            wr_ack_ref = 1'b1;
                        end
                        else begin
                            wr_ack_ref = 1'b0;
                        end
                    end
                    begin
                        if (rd_en && count > 0) begin
                            data_out_ref = mem_ref[rd_ptr];
                            rd_ptr++;
                            empty_ref = (count == 0) ? 1'b1 : 1'b0;
                        end
                    end
                join
                if ({wr_en, rd_en} == 2'b11) begin
                    if (empty_ref)
                        count++;
                    else if (full_ref)
                        count--;
                end
                else if ( ({wr_en, rd_en} == 2'b10) && !full_ref)
                    count++;
                else if ( ({wr_en, rd_en} == 2'b01) && !empty_ref)
                    count--;
                full_ref = (count == FIFO_DEPTH)? 1 : 0;
                empty_ref = (count == 0)? 1 : 0;
                almostfull_ref = (count == FIFO_DEPTH-1)? 1 : 0;
                almostempty_ref = (count == 1)? 1 : 0;
                underflow_ref = (empty_ref)? 1 : 0;
            end
        endfunction
    endclass : FIFO_scoreboard

endpackage
```

Figure 21: Scoreboard Package Code 3

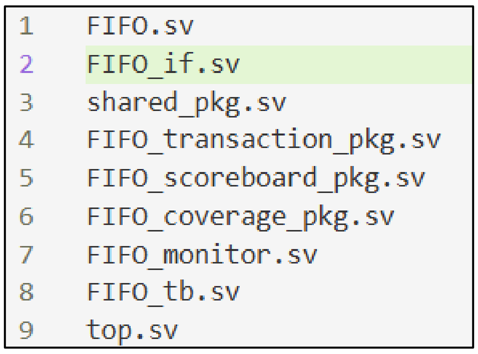
2.10. Shared Package



```
package shared_pkg;
    bit test_finished; // Flag to indicate if the test is finished

    /////////////////////////////////////////////////////////
    ///////////////////// Counters //////////////////////
    /////////////////////////////////////////////////////////

    int correct_count, error_count;

    //////////////////////////////////////////////////////////
    //////////////////// Parameters //////////////////////
    //////////////////////////////////////////////////////////

    parameter int FIFO_WIDTH = 16; // Width of the FIFO data bus
    parameter int FIFO_DEPTH = 8; // Depth of the FIFO
endpackage
```

Figure 22: Shared Package Code

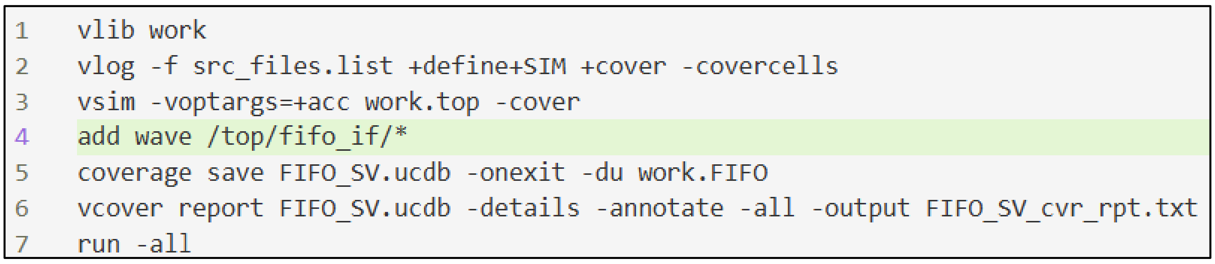
3. Source Files List



```
FIFO.sv
FIFO_if.sv
shared_pkg.sv
FIFO_transaction_pkg.sv
FIFO_scoreboard_pkg.sv
FIFO_coverage_pkg.sv
FIFO_monitor.sv
FIFO_tb.sv
top.sv
```

Figure 23: List of Source Files

4. Do File

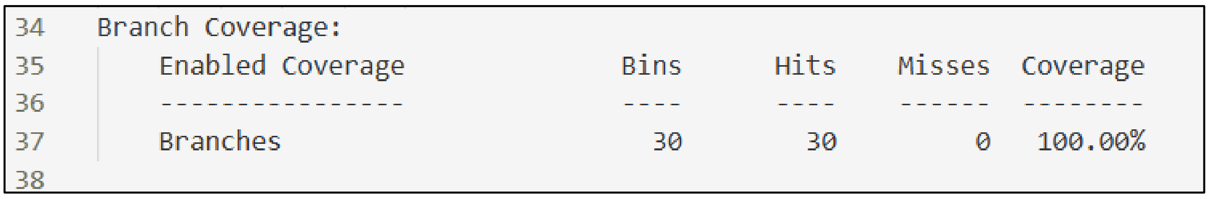


```
vlib work
vlog -f src_files.list +define+SIM +cover -covercells
vsim -voptargs=+acc work.top -cover
add wave /top/fifo_if/*
coverage save FIFO_SV.ucdb -onexit -du work.FIFO
vcover report FIFO_SV.ucdb -details -annotate -all -output FIFO_SV_cvr_rpt.txt
run -all
```

Figure 24: The Do File

5. Coverage Report Snippets

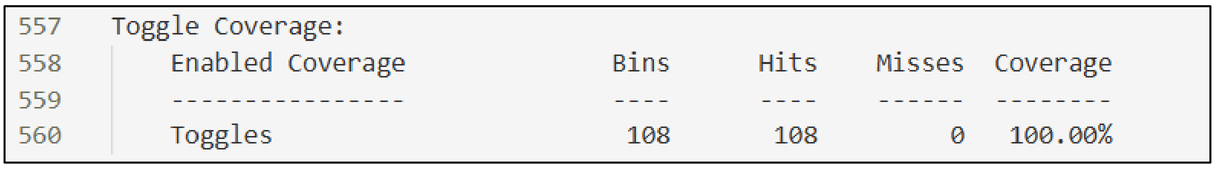
5.1. Branch Coverage Snippet



```
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        30        30         0   100.00%

```

Figure 25: Total Branch Coverage

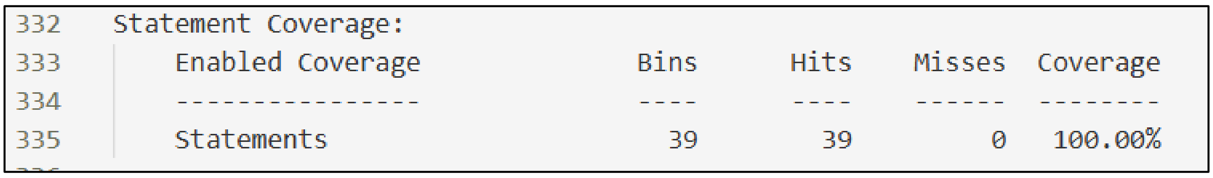
5.2. Toggle Coverage Snippet



```
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                        108       108         0   100.00%
```

Figure 26: Total Toggle Coverage

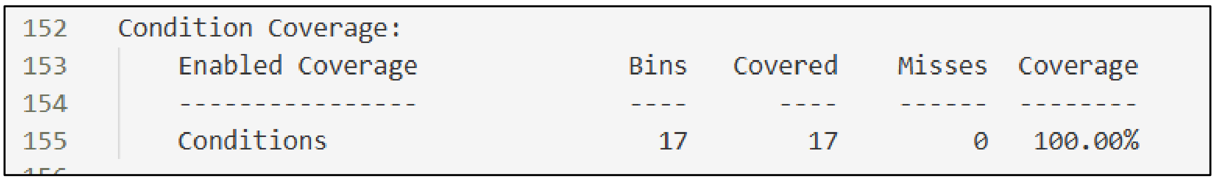
5.3. Statement Coverage Snippet



```
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                      39        39         0   100.00%
```

Figure 27: Total Statement Coverage

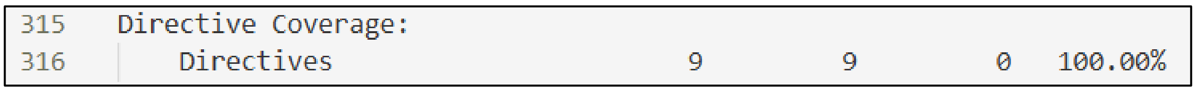
5.4. Condition Coverage Snippet



```
Condition Coverage:
    Enabled Coverage              Bins   Covered    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Conditions                      17        17         0   100.00%
```

Figure 28: Total Condition Coverage

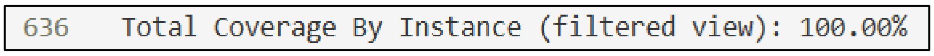
5.5. Directive Coverage Snippet



```
Directive Coverage:
    Directives                       9         9         0   100.00%
```

Figure 29: Total Directive Coverage

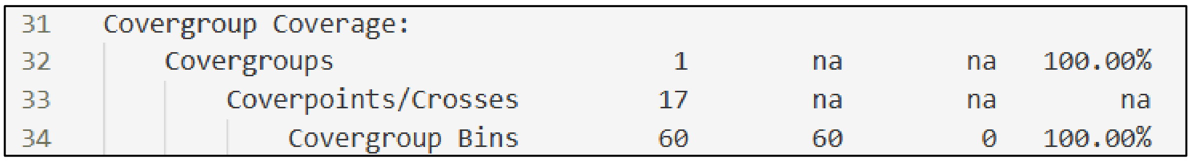
5.6. Total Coverage Snippet



```
Total Coverage By Instance (filtered view): 100.00%
```

Figure 30: Total Coverage

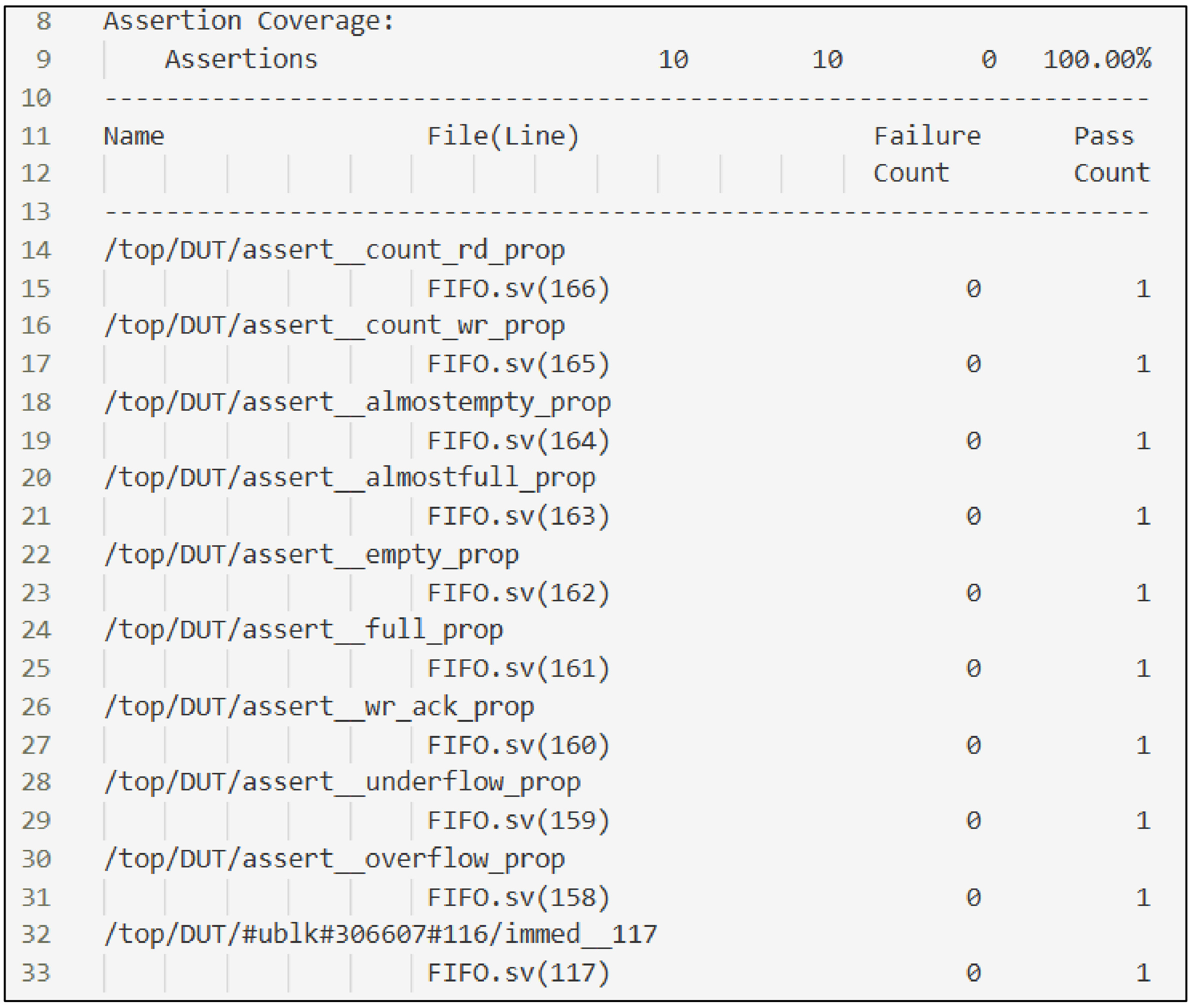
6. Functional Coverage Report Snippet



```
Covergroup Coverage:
    Covergroups                      1        na        na   100.00%
        Coverpoints/Crosses         17        na        na        na
            Covergroup Bins         60        60         0   100.00%
```

Figure 31: Total Functional Coverage

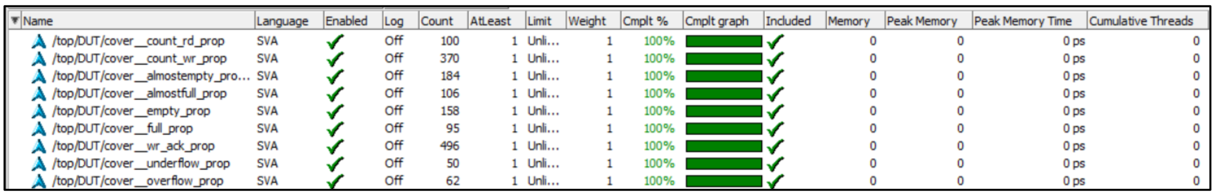
7. Assertions Report Snippet



```
Assertion Coverage:
    Assertions                           10        10          0   100.00%
--------------------------------------------------------------------------
Name                     File(Line)                      Failure      Pass
                                                         Count        Count
--------------------------------------------------------------------------
/top/DUT/assert__count_rd_prop
                         FIFO.sv(166)                          0          1
/top/DUT/assert__count_wr_prop
                         FIFO.sv(165)                          0          1
/top/DUT/assert__almostempty_prop
                         FIFO.sv(164)                          0          1
/top/DUT/assert__almostfull_prop
                         FIFO.sv(163)                          0          1
/top/DUT/assert__empty_prop
                         FIFO.sv(162)                          0          1
/top/DUT/assert__full_prop
                         FIFO.sv(161)                          0          1
/top/DUT/assert__wr_ack_prop
                         FIFO.sv(160)                          0          1
/top/DUT/assert__underflow_prop
                         FIFO.sv(159)                          0          1
/top/DUT/assert__overflow_prop
                         FIFO.sv(158)                          0          1
/top/DUT/#ublk#306607#116/immed__117
                         FIFO.sv(117)                          0          1
```

Figure 32: Assertions

8. Cover Directives Snippet



Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
/top/DUT/cover__count_rd_prop	SVA	✓	Off	100	1	Unli...	1	100%	🟩	✓	0	0	0 ps	0
/top/DUT/cover__count_wr_prop	SVA	✓	Off	370	1	Unli...	1	100%	🟩	✓	0	0	0 ps	0
/top/DUT/cover__almostempty_pro...	SVA	✓	Off	184	1	Unli...	1	100%	🟩	✓	0	0	0 ps	0
/top/DUT/cover__almostfull_prop	SVA	✓	Off	106	1	Unli...	1	100%	🟩	✓	0	0	0 ps	0
/top/DUT/cover__empty_prop	SVA	✓	Off	158	1	Unli...	1	100%	🟩	✓	0	0	0 ps	0
/top/DUT/cover__full_prop	SVA	✓	Off	95	1	Unli...	1	100%	🟩	✓	0	0	0 ps	0
/top/DUT/cover__wr_ack_prop	SVA	✓	Off	496	1	Unli...	1	100%	🟩	✓	0	0	0 ps	0
/top/DUT/cover__underflow_prop	SVA	✓	Off	50	1	Unli...	1	100%	🟩	✓	0	0	0 ps	0
/top/DUT/cover__overflow_prop	SVA	✓	Off	62	1	Unli...	1	100%	🟩	✓	0	0	0 ps	0

Figure 33: Cover Directives