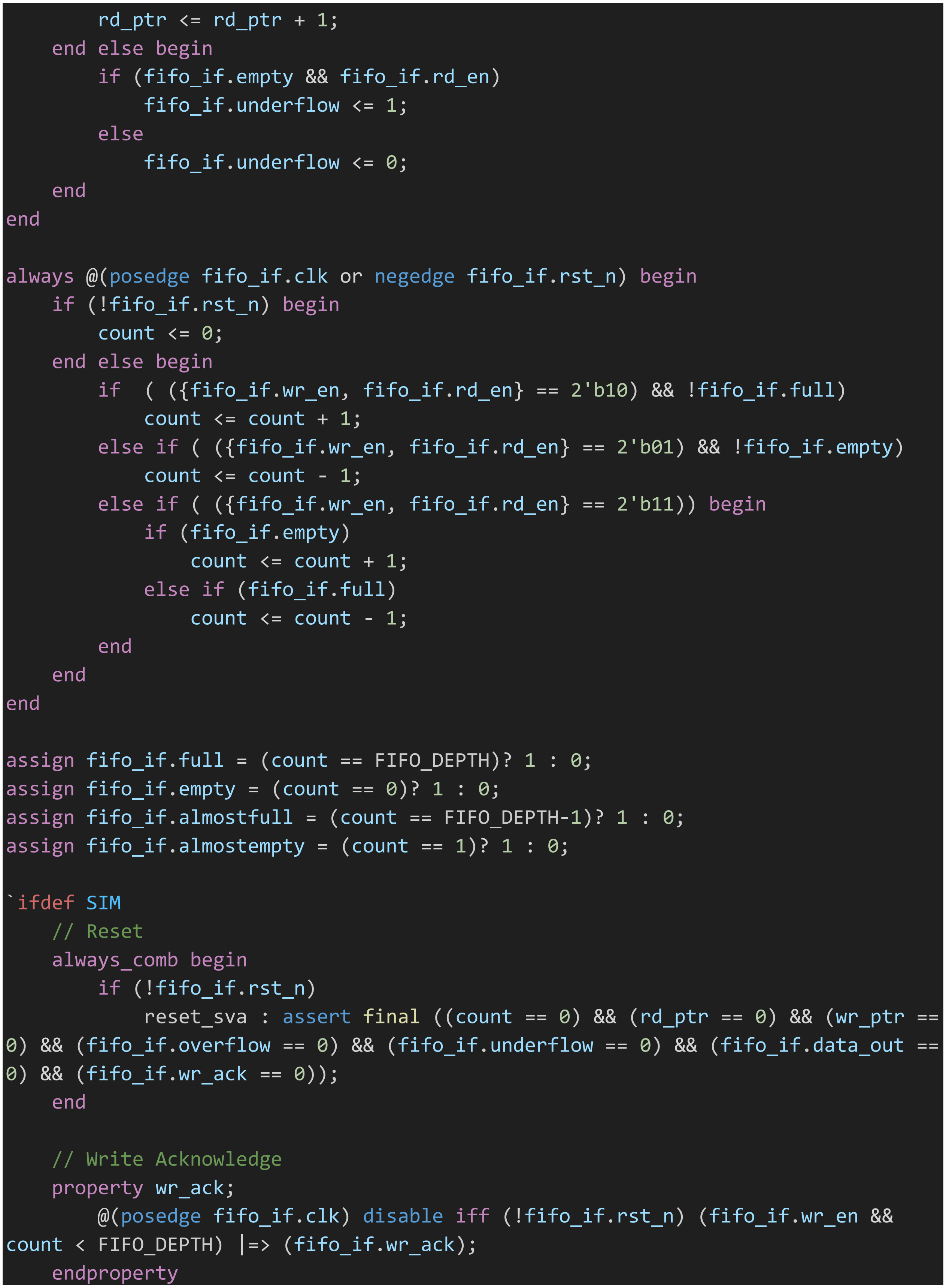
PROJECT 1: FIFO VERIFICATION

For Kareem Waseem

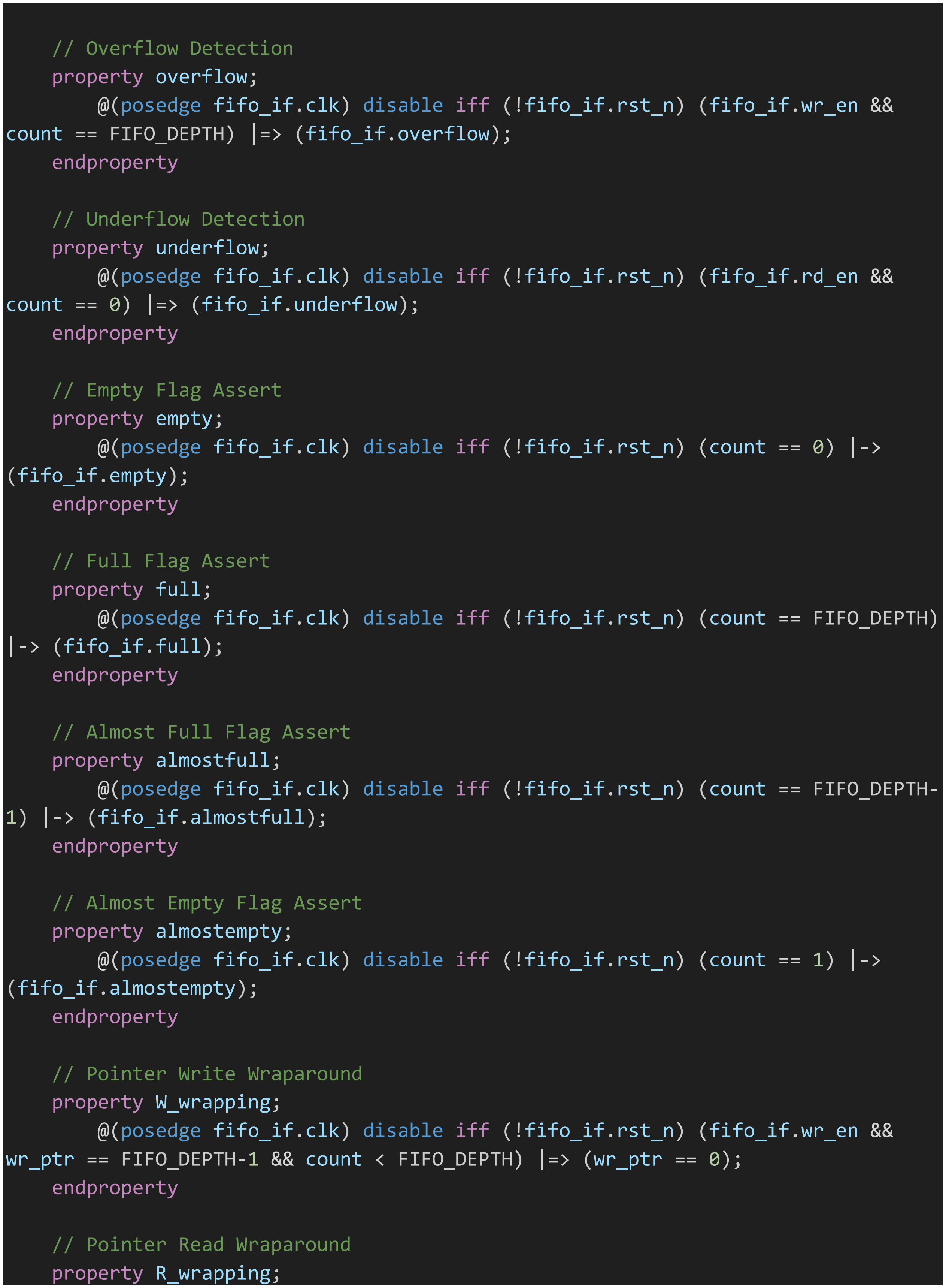
Digital Verification Diploma

Design

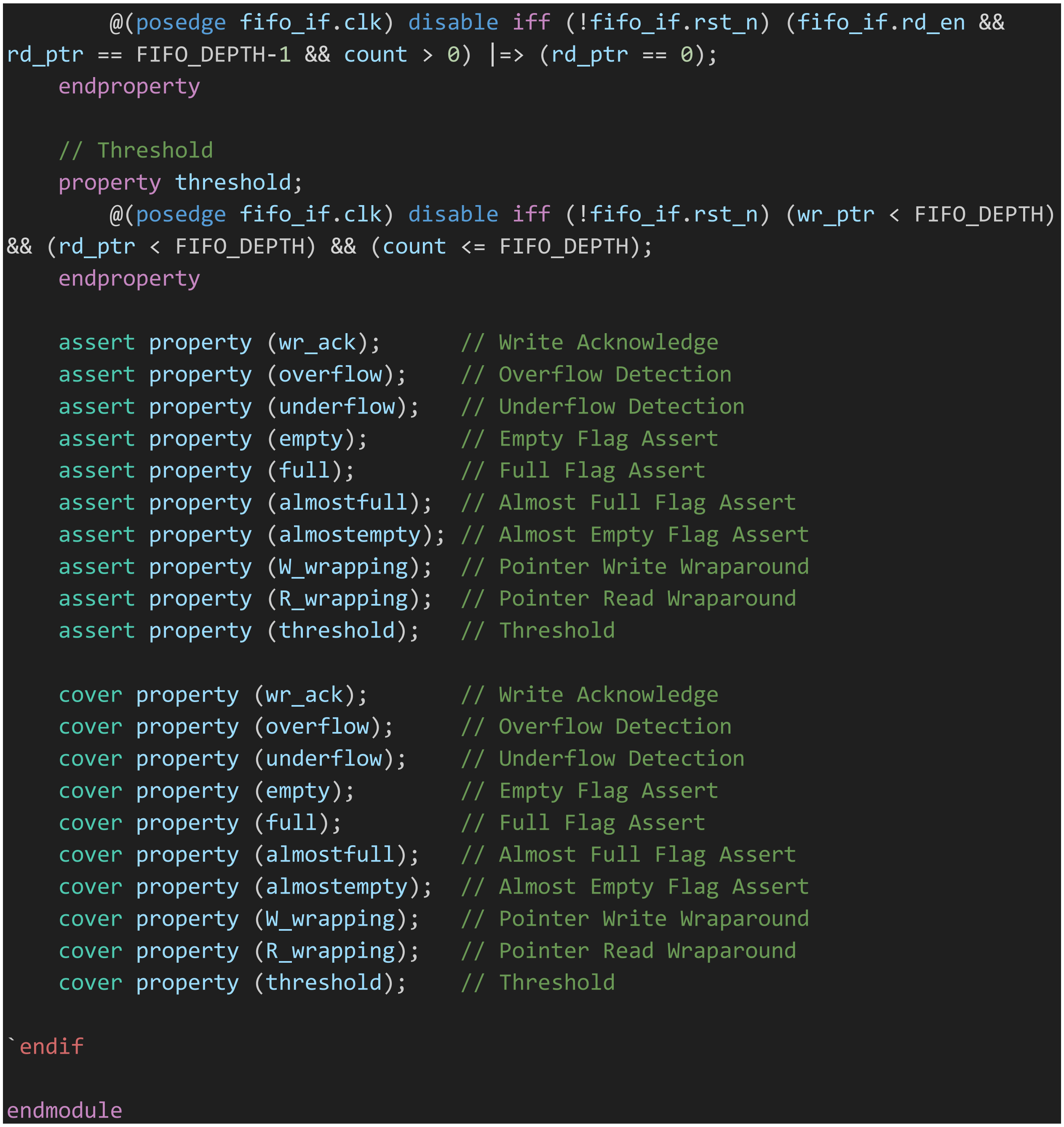
```
////////////////////////////////////////////////////////////////////////////////
// Author: Kareem Waseem
// Course: Digital Verification using SV & UVM
//
// Description: FIFO Design
//
////////////////////////////////////////////////////////////////////////////////
module FIFO (FIFO_if.DUT fifo_if);

parameter FIFO_WIDTH = 16, FIFO_DEPTH = 8;

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 fifo_if.clk or negedge fifo_if.rst_n) begin
    if (!fifo_if.rst_n) begin
        wr_ptr <= 0;
        fifo_if.wr_ack <= 0;
        fifo_if.overflow <= 0;
    end else if (fifo_if.wr_en && count < FIFO_DEPTH) begin
        mem[wr_ptr] <= fifo_if.data_in;
        fifo_if.wr_ack <= 1;
        wr_ptr <= wr_ptr + 1;
    end else begin
        fifo_if.wr_ack <= 0;
        if (fifo_if.full && fifo_if.wr_en)
            fifo_if.overflow <= 1;
        else
            fifo_if.overflow <= 0;
    end
end

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



```
        rd_ptr <= rd_ptr + 1;
    end else begin
        if (fifo_if.empty && fifo_if.rd_en)
            fifo_if.underflow <= 1;
        else
            fifo_if.underflow <= 0;
    end
end

always @(posedge fifo_if.clk or negedge fifo_if.rst_n) begin
    if (!fifo_if.rst_n) begin
        count <= 0;
    end else begin
        if  ( ({fifo_if.wr_en, fifo_if.rd_en} == 2'b10) && !fifo_if.full)
            count <= count + 1;
        else if ( ({fifo_if.wr_en, fifo_if.rd_en} == 2'b01) && !fifo_if.empty)
            count <= count - 1;
        else if ( ({fifo_if.wr_en, fifo_if.rd_en} == 2'b11)) begin
            if (fifo_if.empty)
                count <= count + 1;
            else if (fifo_if.full)
                count <= count - 1;
        end
    end
end

assign fifo_if.full = (count == FIFO_DEPTH)? 1 : 0;
assign fifo_if.empty = (count == 0)? 1 : 0;
assign fifo_if.almostfull = (count == FIFO_DEPTH-1)? 1 : 0;
assign fifo_if.almostempty = (count == 1)? 1 : 0;

`ifdef SIM
    // Reset
    always_comb begin
        if (!fifo_if.rst_n)
            reset_sva : assert final ((count == 0) && (rd_ptr == 0) && (wr_ptr ==
0) && (fifo_if.overflow == 0) && (fifo_if.underflow == 0) && (fifo_if.data_out ==
0) && (fifo_if.wr_ack == 0));
    end

    // Write Acknowledge
    property wr_ack;
        @(posedge fifo_if.clk) disable iff (!fifo_if.rst_n) (fifo_if.wr_en &&
count < FIFO_DEPTH) |=> (fifo_if.wr_ack);
    endproperty
```



```
    // Overflow Detection
    property overflow;
        @(posedge fifo_if.clk) disable iff (!fifo_if.rst_n) (fifo_if.wr_en &&
count == FIFO_DEPTH) |=> (fifo_if.overflow);
    endproperty

    // Underflow Detection
    property underflow;
        @(posedge fifo_if.clk) disable iff (!fifo_if.rst_n) (fifo_if.rd_en &&
count == 0) |=> (fifo_if.underflow);
    endproperty

    // Empty Flag Assert
    property empty;
        @(posedge fifo_if.clk) disable iff (!fifo_if.rst_n) (count == 0) |->
(fifo_if.empty);
    endproperty

    // Full Flag Assert
    property full;
        @(posedge fifo_if.clk) disable iff (!fifo_if.rst_n) (count == FIFO_DEPTH)
|-> (fifo_if.full);
    endproperty

    // Almost Full Flag Assert
    property almostfull;
        @(posedge fifo_if.clk) disable iff (!fifo_if.rst_n) (count == FIFO_DEPTH-
1) |-> (fifo_if.almostfull);
    endproperty

    // Almost Empty Flag Assert
    property almostempty;
        @(posedge fifo_if.clk) disable iff (!fifo_if.rst_n) (count == 1) |->
(fifo_if.almostempty);
    endproperty

    // Pointer Write Wraparound
    property W_wrapping;
        @(posedge fifo_if.clk) disable iff (!fifo_if.rst_n) (fifo_if.wr_en &&
wr_ptr == FIFO_DEPTH-1 && count < FIFO_DEPTH) |=> (wr_ptr == 0);
    endproperty

    // Pointer Read Wraparound
    property R_wrapping;
```



```
        @(posedge fifo_if.clk) disable iff (!fifo_if.rst_n) (fifo_if.rd_en &&
rd_ptr == FIFO_DEPTH-1 && count > 0) |=> (rd_ptr == 0);
    endproperty

    // Threshold
    property threshold;
        @(posedge fifo_if.clk) disable iff (!fifo_if.rst_n) (wr_ptr < FIFO_DEPTH)
&& (rd_ptr < FIFO_DEPTH) && (count <= FIFO_DEPTH);
    endproperty

    assert property (wr_ack);      // Write Acknowledge
    assert property (overflow);    // Overflow Detection
    assert property (underflow);   // Underflow Detection
    assert property (empty);       // Empty Flag Assert
    assert property (full);        // Full Flag Assert
    assert property (almostfull);  // Almost Full Flag Assert
    assert property (almostempty); // Almost Empty Flag Assert
    assert property (W_wrapping);  // Pointer Write Wraparound
    assert property (R_wrapping);  // Pointer Read Wraparound
    assert property (threshold);   // Threshold

    cover property (wr_ack);       // Write Acknowledge
    cover property (overflow);     // Overflow Detection
    cover property (underflow);    // Underflow Detection
    cover property (empty);        // Empty Flag Assert
    cover property (full);         // Full Flag Assert
    cover property (almostfull);   // Almost Full Flag Assert
    cover property (almostempty);  // Almost Empty Flag Assert
    cover property (W_wrapping);   // Pointer Write Wraparound
    cover property (R_wrapping);   // Pointer Read Wraparound
    cover property (threshold);    // Threshold

`endif

endmodule
```

- Handle reset as it was only reset pointers and counter.
- Make underflow flag inside always block as this flag is sequential (as specs said).
- Make almostfull flag high when counter is less than the full by 1 not by 2 (as specs said).
- Make the condition of overflow flag with (&&) as it was (&).
- Adding the missing case as I can read and write at the same time so we have to handle it when updating counter of the design.

Verification Plan

Label	Design Required Description	Stimulus Generation	Functional Coverage	Functionality Check
FIFO Reset Test	When rst_n is low, all outputs must reset (data_out=0, wr_ack=0, underflow=0, overflow=0)	Directed at the start of the simulation	reset_cp coverpoint (bins rst_0, rst_1)	reset_sva assertion to make sure output is correct
FIFO Write Test	Verify writing in fifo with random inputs	Directed by using repeat block during the simulation	wr_en_cp, wr_ack_F_cp, overflow_F_cp, full_F_cp, almostfull_F_cp (cross coverage with wr_en)	Assertions: wr_ack, overflow, full, almostfull to make sure output is correct
FIFO Read Test	Verify reading when data exists and underflow detection when queue empty	Directed by using repeat block during the simulation	rd_en_cp, underflow_F_cp, empty_F_cp, almostempty_F_cp (cross coverage with rd_en)	Assertions: underflow, empty, almostempty; scoreboard check_data ensures popped values match golden queue
FIFO Write and Read Test	Verify simultaneous read and write updates pointers/count correctly	Directed by using repeat block during the simulation	Cross coverpoints: x_wr_rd_full, x_wr_rd_empty, x_wr_rd_af, x_wr_rd_ae, x_wr_rd_ack, x_wr_rd_of, x_wr_rd_uf → ensures corner cases of simultaneous ops	Assertions: wr_ack, underflow, overflow, W_wrapping, R_wrapping, threshold; scoreboard check_data ensures correct enqueue+dequeue behavior
FIFO General Test	Randomized operations must match scoreboard reference model for all input combinations	Randomized inside repeat block during the simulation	All coverpoints + crosses: wr_en_cp, rd_en_cp, flag coverpoints (full, empty, almostfull, almostempty, overflow, underflow, wr_ack)	Assertions + scoreboard check_data: ensures DUT matches reference model across random scenarios, hitting all corner cases and achieving full functional coverage goals.

Design Verification

Shared Package

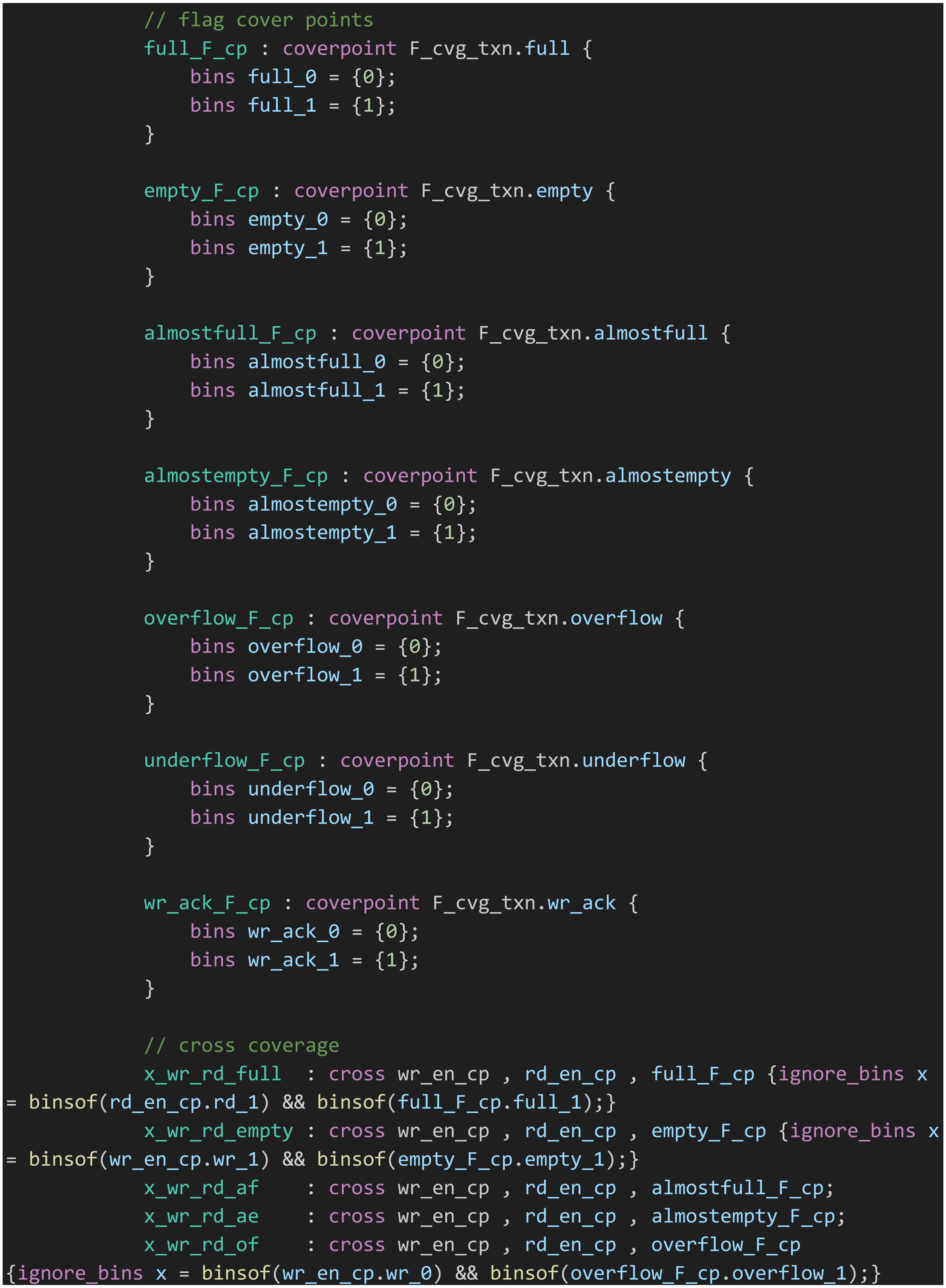
```
package shared_pkg;

    bit test_finished;
    int correct_count = 0, error_count = 0;

endpackage : shared_pkg
```

FIFO Transaction

```
package FIFO_transaction_pkg;
    import shared_pkg::*;

    class FIFO_transaction;

        logic [15:0] data_out;
        logic wr_ack, overflow, full, empty, almostfull, almostempty, underflow;

        int RD_EN_ON_DIST, WR_EN_ON_DIST;
        rand logic rst_n, wr_en, rd_en;
        rand logic [15:0] data_in;

        function new(int rd_dist = 30, int wr_dist = 70);
            RD_EN_ON_DIST = rd_dist;
            WR_EN_ON_DIST = wr_dist;
        endfunction : new

        constraint reset_c {rst_n dist {1 := 98 , 0 := 2};;}
        constraint wr_en_c {wr_en dist {1 := WR_EN_ON_DIST , 0 := (100 -
WR_EN_ON_DIST)};;}
        constraint rd_en_c {rd_en dist {1 := RD_EN_ON_DIST , 0 := (100 -
RD_EN_ON_DIST)};;}

    endclass : FIFO_transaction

endpackage : FIFO_transaction_pkg
```

FIFO Coverage

```
package FIFO_coverage_pkg;
    import FIFO_transaction_pkg::*;

    class FIFO_coverage;

        FIFO_transaction F_cvg_txn;

        covergroup cg;
            wr_en_cp : coverpoint F_cvg_txn.wr_en {
                bins wr_0 = {0};
                bins wr_1 = {1};
            }

            rd_en_cp : coverpoint F_cvg_txn.rd_en {
                bins rd_0 = {0};
                bins rd_1 = {1};
            }

            reset_cp : coverpoint F_cvg_txn.rst_n {
                bins rst_0 = {0};
                bins rst_1 = {1};
            }
```



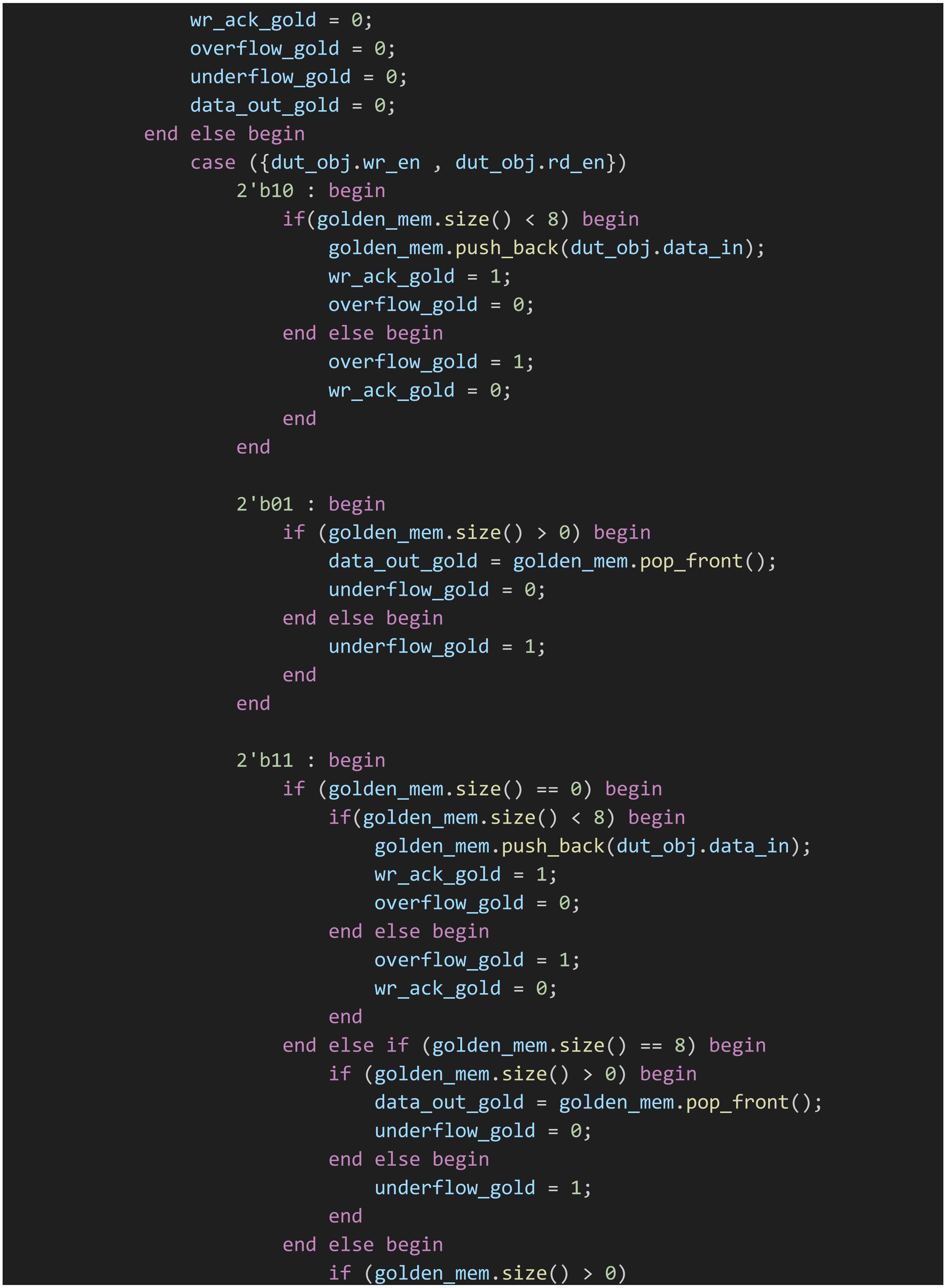
```
            // flag cover points
            full_F_cp : coverpoint F_cvg_txn.full {
                bins full_0 = {0};
                bins full_1 = {1};
            }

            empty_F_cp : coverpoint F_cvg_txn.empty {
                bins empty_0 = {0};
                bins empty_1 = {1};
            }

            almostfull_F_cp : coverpoint F_cvg_txn.almostfull {
                bins almostfull_0 = {0};
                bins almostfull_1 = {1};
            }

            almostempty_F_cp : coverpoint F_cvg_txn.almostempty {
                bins almostempty_0 = {0};
                bins almostempty_1 = {1};
            }

            overflow_F_cp : coverpoint F_cvg_txn.overflow {
                bins overflow_0 = {0};
                bins overflow_1 = {1};
            }

            underflow_F_cp : coverpoint F_cvg_txn.underflow {
                bins underflow_0 = {0};
                bins underflow_1 = {1};
            }

            wr_ack_F_cp : coverpoint F_cvg_txn.wr_ack {
                bins wr_ack_0 = {0};
                bins wr_ack_1 = {1};
            }

            // cross coverage
            x_wr_rd_full  : cross wr_en_cp , rd_en_cp , full_F_cp {ignore_bins x
= binsof(rd_en_cp.rd_1) && binsof(full_F_cp.full_1);}
            x_wr_rd_empty : cross wr_en_cp , rd_en_cp , empty_F_cp {ignore_bins x
= binsof(wr_en_cp.wr_1) && binsof(empty_F_cp.empty_1);}
            x_wr_rd_af    : cross wr_en_cp , rd_en_cp , almostfull_F_cp;
            x_wr_rd_ae    : cross wr_en_cp , rd_en_cp , almostempty_F_cp;
            x_wr_rd_of    : cross wr_en_cp , rd_en_cp , overflow_F_cp
{ignore_bins x = binsof(wr_en_cp.wr_0) && binsof(overflow_F_cp.overflow_1);}
```

```
            x_wr_rd_uf    : cross wr_en_cp , rd_en_cp , underflow_F_cp
{ignore_bins x = binsof(rd_en_cp.rd_0) && binsof(underflow_F_cp.underflow_1);}
            x_wr_rd_ack   : cross wr_en_cp , rd_en_cp , wr_ack_F_cp {ignore_bins
x = binsof(wr_en_cp.wr_0) && binsof(wr_ack_F_cp.wr_ack_1);}

        endgroup : cg

        function new();
            cg = new();
        endfunction : new

        function void sample_data (FIFO_transaction F_txn);
            F_cvg_txn = F_txn;
            cg.sample();
        endfunction : sample_data

    endclass : FIFO_coverage

endpackage : FIFO_coverage_pkg
```

- ○ The ignored bins are for the cases that could not be hit or be achieved:
- Full on with rd_en on as full is combinational so on seeing read operation it changed immediately and there is no meaning to see full flag with read operation.
- Empty on with wr_en on as empty is combinational so on seeing write operation it changed immediately and there is no meaning to see empty flag with write operation.
- The overflow on with wr_en off as overflow is defined signal to detect attempt to write for full FIFO so it can’t be achieved.
- The underflow on with rd_en off as underflow is defined signal to detect attempt to read for empty FIFO so it can’t be achieved.
- The wr_ack on with wr_en off as wr_ack is defined to sure that the write operation is done so it can’t be achieved.

FIFO Scoreboard

```
package FIFO_scoreboard_pkg;
    import FIFO_transaction_pkg::*;
    import shared_pkg::*;

    class FIFO_scoreboard;

        bit [15:0] data_out_gold;
        bit wr_ack_gold, overflow_gold, full_gold, empty_gold, almostfull_gold, almostempty_gold, underflow_gold;

        logic [15:0] golden_mem [$]; // queue for golden model

        function void check_data (FIFO_transaction obj);
            reference_model(obj);

            // compare data_out only when read happened and no underflow
            if (obj.rd_en && golden_mem.size() >= 0) begin
                if (underflow_gold) begin
                    if (!obj.underflow) begin
                        $display("ERROR: DUT underflow not asserted when expected");
                        error_count++;
                    end else
                        correct_count++;
                end else begin
                    if (data_out_gold == obj.data_out)
                        correct_count++;
                    else begin
                        $display("MISMATCH: DUT data out is %0h and expected %0h",obj.data_out,data_out_gold);
                        error_count++;
                    end
                end
            end
        endfunction : check_data

        function void reference_model (FIFO_transaction dut_obj);
            wr_ack_gold    = 0;
            overflow_gold = 0;
            underflow_gold = 0;

            if (!dut_obj.rst_n) begin
                golden_mem.delete();
```

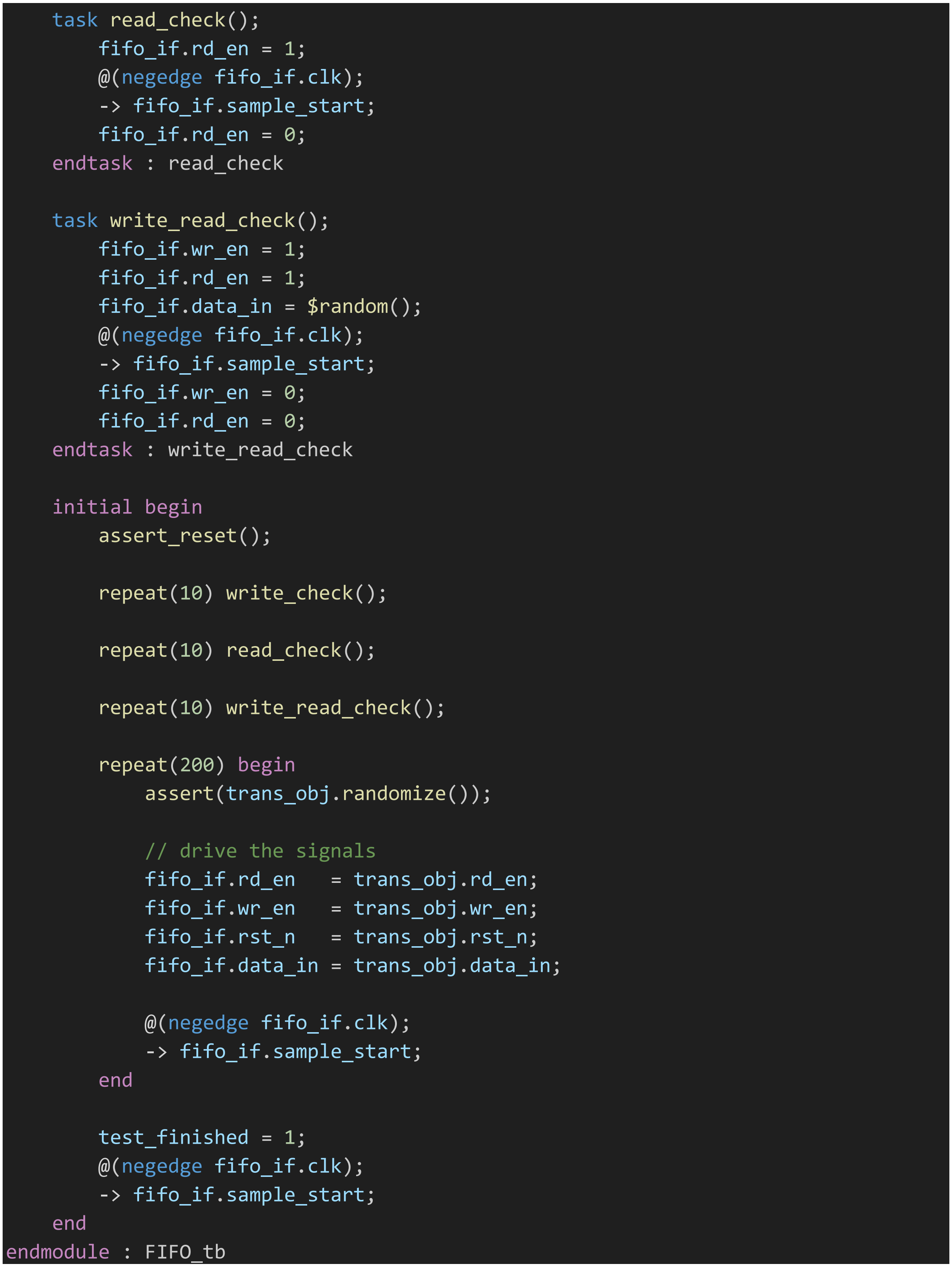


```
                wr_ack_gold = 0;
                overflow_gold = 0;
                underflow_gold = 0;
                data_out_gold = 0;
            end else begin
                case ({dut_obj.wr_en , dut_obj.rd_en})
                    2'b10 : begin
                        if(golden_mem.size() < 8) begin
                            golden_mem.push_back(dut_obj.data_in);
                            wr_ack_gold = 1;
                            overflow_gold = 0;
                        end else begin
                            overflow_gold = 1;
                            wr_ack_gold = 0;
                        end
                    end

                    2'b01 : begin
                        if (golden_mem.size() > 0) begin
                            data_out_gold = golden_mem.pop_front();
                            underflow_gold = 0;
                        end else begin
                            underflow_gold = 1;
                        end
                    end

                    2'b11 : begin
                        if (golden_mem.size() == 0) begin
                            if(golden_mem.size() < 8) begin
                                golden_mem.push_back(dut_obj.data_in);
                                wr_ack_gold = 1;
                                overflow_gold = 0;
                            end else begin
                                overflow_gold = 1;
                                wr_ack_gold = 0;
                            end
                        end else if (golden_mem.size() == 8) begin
                            if (golden_mem.size() > 0) begin
                                data_out_gold = golden_mem.pop_front();
                                underflow_gold = 0;
                            end else begin
                                underflow_gold = 1;
                            end
                        end else begin
                            if (golden_mem.size() > 0)
```

```
                                data_out_gold = golden_mem.pop_front();

                            if (golden_mem.size() < 8) begin
                                golden_mem.push_back(dut_obj.data_in);
                                wr_ack_gold = 1;
                            end
                        end
                    end
                endcase

                // flag update
                empty_gold = (golden_mem.size() == 0);
                full_gold = (golden_mem.size() == 8);
                almostempty_gold = (golden_mem.size() == 1);
                almostfull_gold = (golden_mem.size() == 7);
            end
        endfunction : reference_model

    endclass : FIFO_scoreboard

endpackage : FIFO_scoreboard_pkg
```

FIFO Testbench

```
module FIFO_tb (FIFO_if.TEST fifo_if);
    import shared_pkg::*;
    import FIFO_transaction_pkg::*;

    FIFO_transaction trans_obj = new();

    task assert_reset();
        fifo_if.rst_n = 0;
        @(negedge fifo_if.clk);
        -> fifo_if.sample_start;
        fifo_if.rst_n = 1;
    endtask : assert_reset

    task write_check();
        fifo_if.wr_en = 1;
        fifo_if.rd_en = 0;
        fifo_if.data_in = $random();
        @(negedge fifo_if.clk);
        -> fifo_if.sample_start;
        fifo_if.wr_en = 0;
    endtask : write_check
```



```
    task read_check();
        fifo_if.rd_en = 1;
        @(negedge fifo_if.clk);
        -> fifo_if.sample_start;
        fifo_if.rd_en = 0;
    endtask : read_check

    task write_read_check();
        fifo_if.wr_en = 1;
        fifo_if.rd_en = 1;
        fifo_if.data_in = $random();
        @(negedge fifo_if.clk);
        -> fifo_if.sample_start;
        fifo_if.wr_en = 0;
        fifo_if.rd_en = 0;
    endtask : write_read_check

    initial begin
        assert_reset();

        repeat(10) write_check();

        repeat(10) read_check();

        repeat(10) write_read_check();

        repeat(200) begin
            assert(trans_obj.randomize());

            // drive the signals
            fifo_if.rd_en   = trans_obj.rd_en;
            fifo_if.wr_en   = trans_obj.wr_en;
            fifo_if.rst_n   = trans_obj.rst_n;
            fifo_if.data_in = trans_obj.data_in;

            @(negedge fifo_if.clk);
            -> fifo_if.sample_start;
        end

        test_finished = 1;
        @(negedge fifo_if.clk);
        -> fifo_if.sample_start;
    end
endmodule : FIFO_tb
```

Monitor

```
module FIFO_monitor (FIFO_if.MON fifo_if);
    import FIFO_transaction_pkg::*;
    import FIFO_coverage_pkg::*;
    import FIFO_scoreboard_pkg::*;
    import shared_pkg::*;

    FIFO_transaction trans_obj = new();
    FIFO_coverage cov_obj = new();
    FIFO_scoreboard sb_obj = new();

    initial begin
        forever begin
            @(fifo_if.sample_start);
            @(negedge fifo_if.clk);
            trans_obj.data_in      = fifo_if.data_in;
            trans_obj.rst_n        = fifo_if.rst_n;
            trans_obj.wr_en        = fifo_if.wr_en;
            trans_obj.rd_en        = fifo_if.rd_en;
            trans_obj.data_out     = fifo_if.data_out;
            trans_obj.wr_ack       = fifo_if.wr_ack;
            trans_obj.full         = fifo_if.full;
            trans_obj.empty        = fifo_if.empty;
            trans_obj.almostfull   = fifo_if.almostfull;
            trans_obj.almostempty = fifo_if.almostempty;
            trans_obj.overflow     = fifo_if.overflow;
            trans_obj.underflow    = fifo_if.underflow;

            fork
                begin
                    cov_obj.sample_data(trans_obj);
                end
                begin
                    sb_obj.check_data(trans_obj);
                end
            join
            if (test_finished) begin
                $display("Simulation finished. correct_count =%0d error_count
=%0d", correct_count, error_count);
                $stop();
            end
        end
    end
endmodule : FIFO_monitor
```

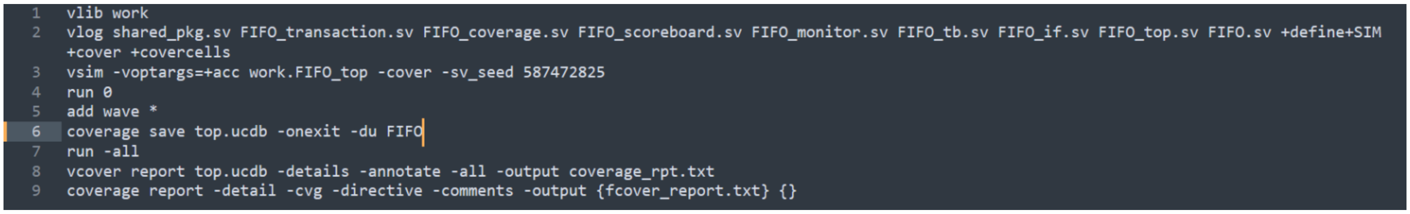
Interface

```
interface FIFO_if (clk);

    localparam FIFO_WIDTH = 16, FIFO_DEPTH = 8;

    input bit clk;

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

    event sample_start;

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

endinterface : FIFO_if
```

Top

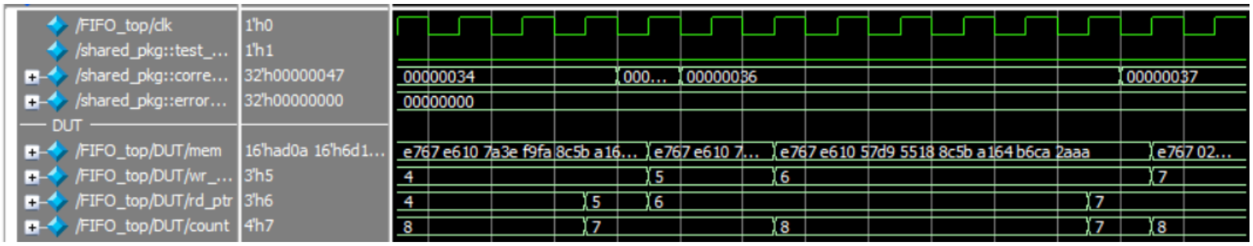
```
module FIFO_top ();

    parameter FIFO_WIDTH = 16 , FIFO_DEPTH = 8;

    bit clk;

    initial begin
        forever #5 clk = ~clk;
    end

    // instantiate
    FIFO_if fifo_if (clk);

    FIFO DUT (fifo_if);
    FIFO_monitor MON (fifo_if);
    FIFO_tb TEST (fifo_if);
endmodule : FIFO_top
```

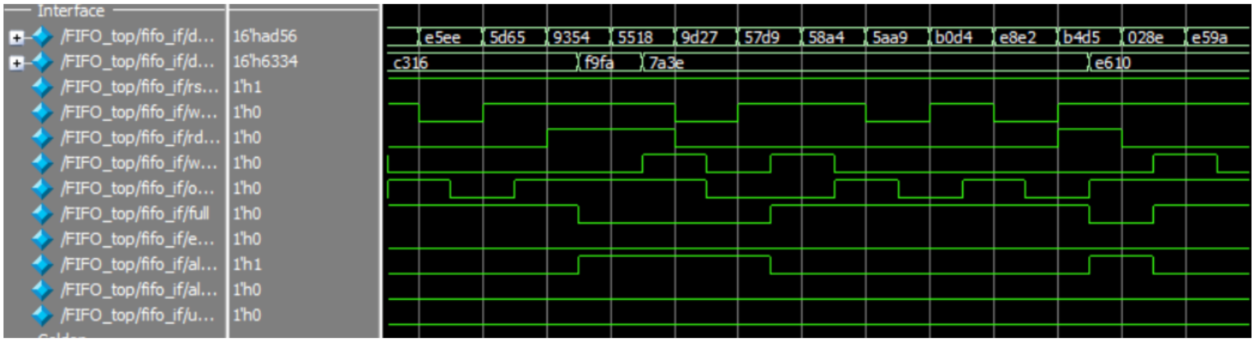
Do file

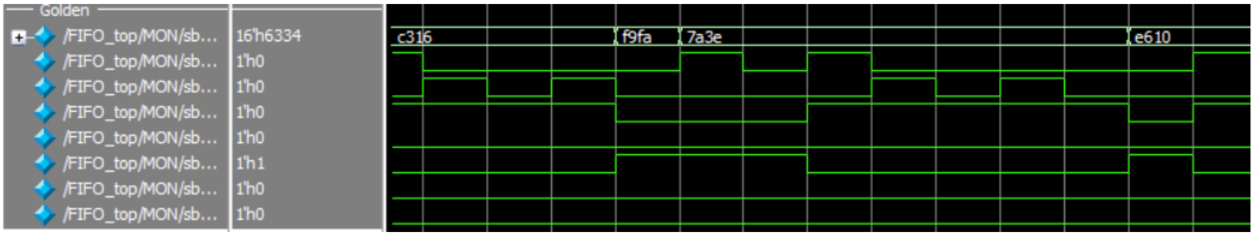


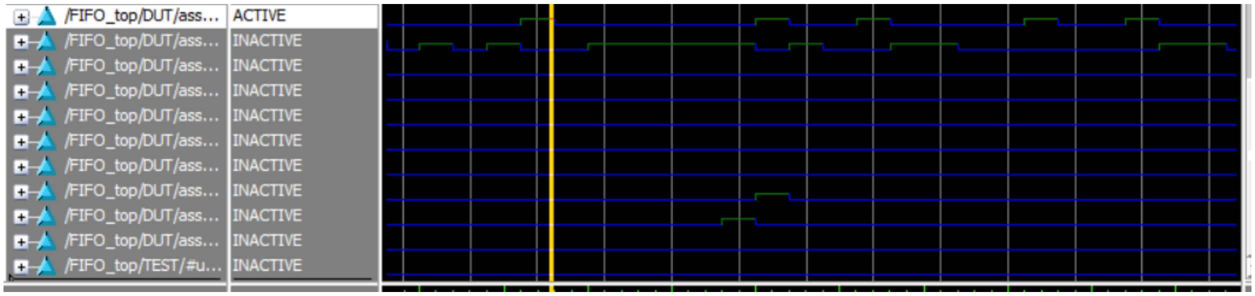
```
vlib work
vlog shared_pkg.sv FIFO_transaction.sv FIFO_coverage.sv FIFO_scoreboard.sv FIFO_monitor.sv FIFO_tb.sv FIFO_if.sv FIFO_top.sv FIFO.sv +define+SIM +cover +covercells
vsim -voptargs=+acc work.FIFO_top -cover -sv_seed 587472825
run 0
add wave *
coverage save top.ucdb -onexit -du FIFO
run -all
vcover report top.ucdb -details -annotate -all -output coverage_rpt.txt
coverage report -detail -cvg -directive -comments -output {fcover_report.txt} {}
```

Waveform

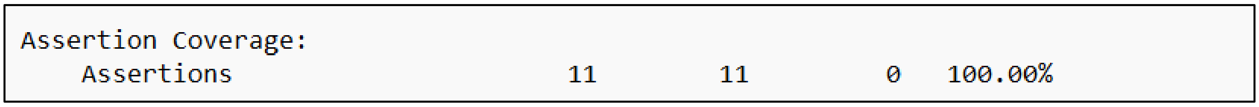

/FIFO_top/clk 1'h0
/shared_pkg::test_... 1'h1
/shared_pkg::corre... 32'h00000047
/shared_pkg::error... 32'h00000000
DUT
/FIFO_top/DUT/mem 16'had0a 16'h6d1...
/FIFO_top/DUT/wr_... 3'h5
/FIFO_top/DUT/rd_ptr 3'h6
/FIFO_top/DUT/count 4'h7
00000034
000...
00000036
00000037
00000000
e767 e610 7a3e f9fa 8c5b a16...
e767 e610 7...
e767 e610 57d9 5518 8c5b a164 b6ca 2aaa
e767 02...
4
5
6
7
4
5
6
7
8
7
8
7
8


Interface
/FIFO_top/fifo_if/d... 16'had56
/FIFO_top/fifo_if/d... 16'h6334
/FIFO_top/fifo_if/rs... 1'h1
/FIFO_top/fifo_if/w... 1'h0
/FIFO_top/fifo_if/rd... 1'h0
/FIFO_top/fifo_if/w... 1'h0
/FIFO_top/fifo_if/o... 1'h0
/FIFO_top/fifo_if/full 1'h0
/FIFO_top/fifo_if/e... 1'h0
/FIFO_top/fifo_if/al... 1'h1
/FIFO_top/fifo_if/al... 1'h0
/FIFO_top/fifo_if/u... 1'h0
e5ee
5d65
9354
5518
9d27
57d9
58a4
5aa9
b0d4
e8e2
b4d5
028e
e59a
c316
f9fa
7a3e
e610

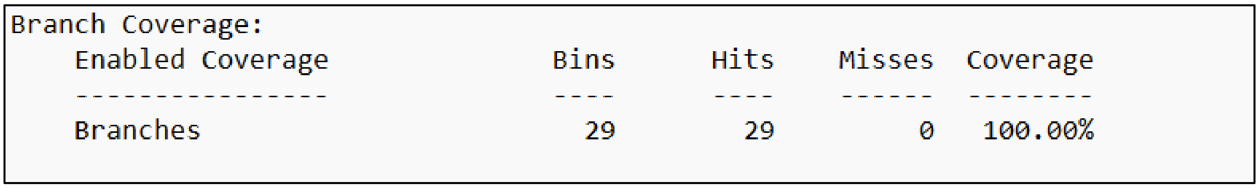



/FIFO_top/DUT/ass... ACTIVE
/FIFO_top/DUT/ass... INACTIVE
/FIFO_top/DUT/ass... INACTIVE
/FIFO_top/DUT/ass... INACTIVE
/FIFO_top/DUT/ass... INACTIVE
/FIFO_top/DUT/ass... INACTIVE
/FIFO_top/DUT/ass... INACTIVE
/FIFO_top/DUT/ass... INACTIVE
/FIFO_top/DUT/ass... INACTIVE
/FIFO_top/DUT/ass... INACTIVE
/FIFO_top/TEST/#u... INACTIVE

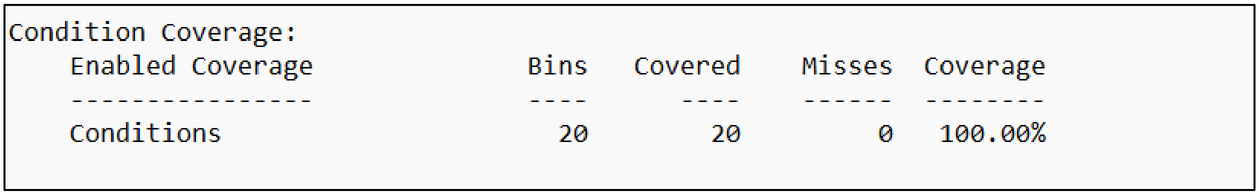
Coverage Report



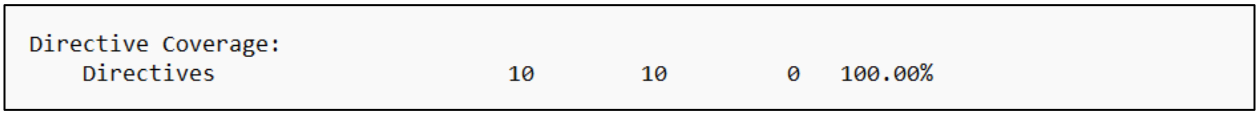
```
Assertion Coverage:
    Assertions                         11         11          0    100.00%
```



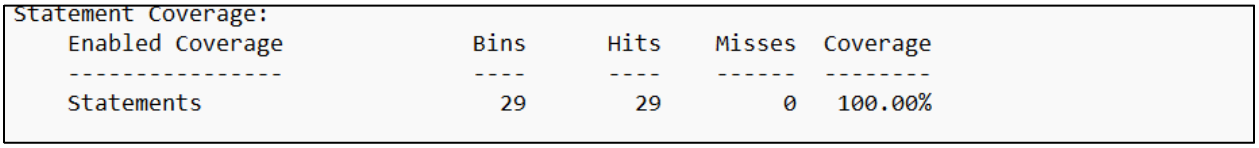
```
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        29        29         0   100.00%
```



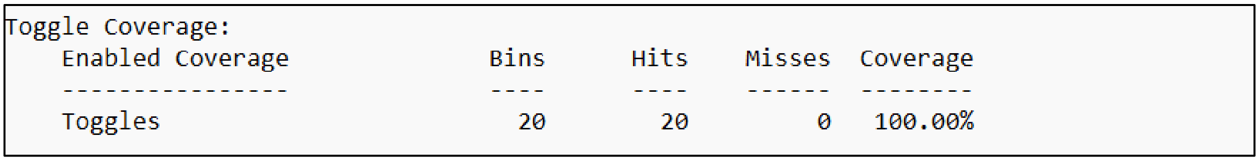
```
Condition Coverage:
    Enabled Coverage              Bins   Covered    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Conditions                      20        20         0   100.00%
```



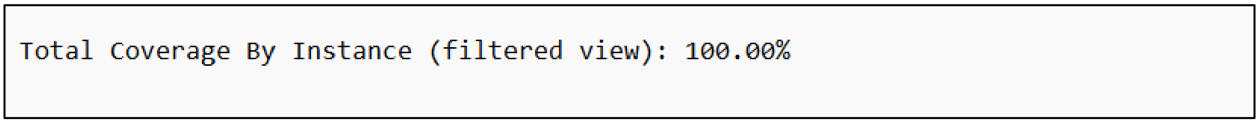
```
Directive Coverage:
    Directives                    10         10          0    100.00%
```



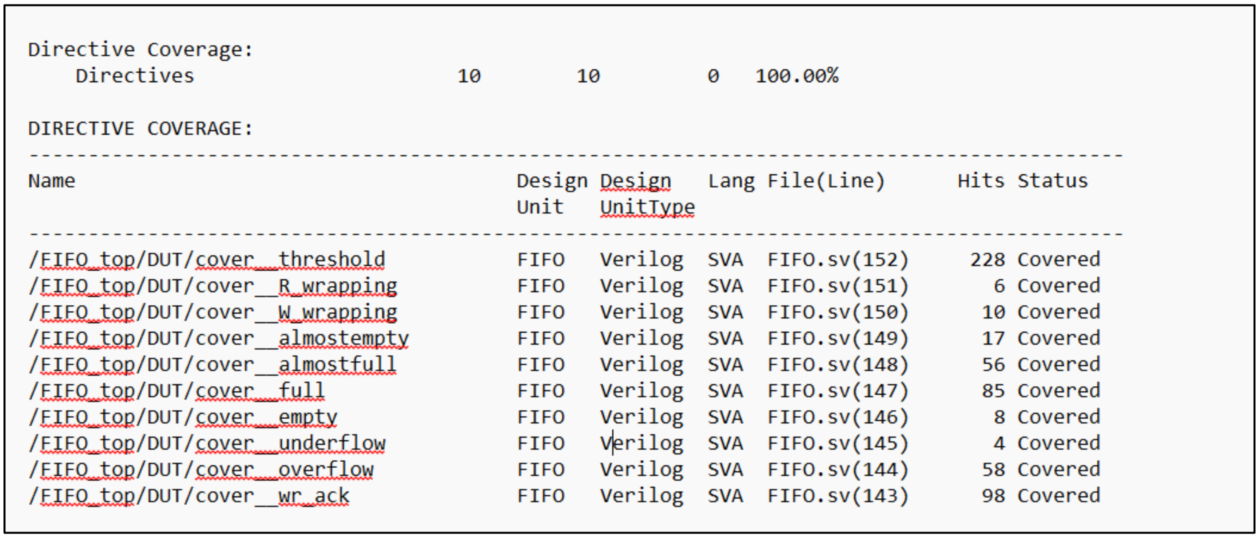
```
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                      29        29         0   100.00%
```



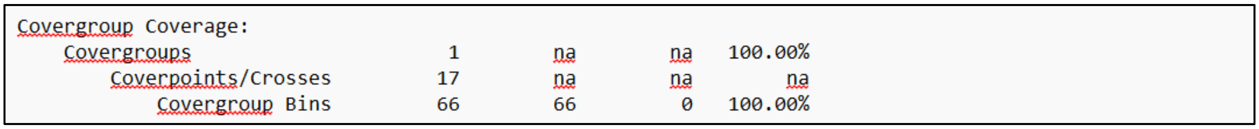
```
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         20        20         0   100.00%
```



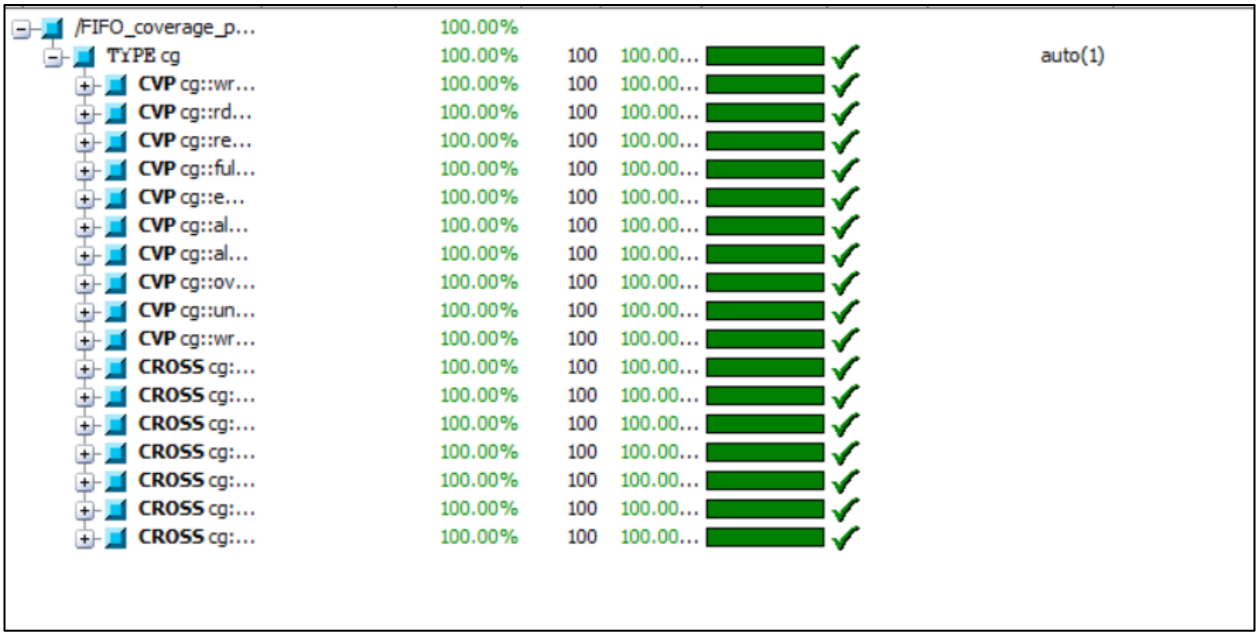
```
Total Coverage By Instance (filtered view): 100.00%
```



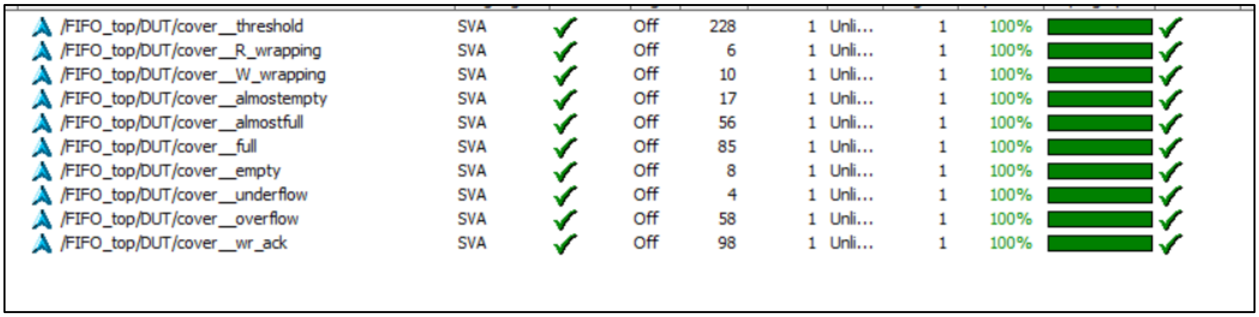
```
Directive Coverage:
    Directives                    10         10          0    100.00%

DIRECTIVE COVERAGE:
--------------------------------------------------------------------------------------------
Name                                     Design Design   Lang File(Line)      Hits Status
                                         Unit   UnitType
--------------------------------------------------------------------------------------------
/FIFO_top/DUT/cover__threshold           FIFO   Verilog  SVA  FIFO.sv(152)     228 Covered
/FIFO_top/DUT/cover__R_wrapping          FIFO   Verilog  SVA  FIFO.sv(151)       6 Covered
/FIFO_top/DUT/cover__W_wrapping          FIFO   Verilog  SVA  FIFO.sv(150)      10 Covered
/FIFO_top/DUT/cover__almostempty         FIFO   Verilog  SVA  FIFO.sv(149)      17 Covered
/FIFO_top/DUT/cover__almostfull          FIFO   Verilog  SVA  FIFO.sv(148)      56 Covered
/FIFO_top/DUT/cover__full                FIFO   Verilog  SVA  FIFO.sv(147)      85 Covered
/FIFO_top/DUT/cover__empty               FIFO   Verilog  SVA  FIFO.sv(146)       8 Covered
/FIFO_top/DUT/cover__underflow           FIFO   Verilog  SVA  FIFO.sv(145)       4 Covered
/FIFO_top/DUT/cover__overflow            FIFO   Verilog  SVA  FIFO.sv(144)      58 Covered
/FIFO_top/DUT/cover__wr_ack              FIFO   Verilog  SVA  FIFO.sv(143)      98 Covered
```



```
Covergroup Coverage:
    Covergroups                      1         na         na    100.00%
        Coverpoints/Crosses         17         na         na         na
            Covergroup Bins         66         66          0    100.00%
```



```
⊟ /FIFO_coverage_p...        100.00%
  ⊟ TYPE cg                  100.00%    100   100.00...  ✓        auto(1)
    ⊞ CVP cg::wr...          100.00%    100   100.00...  ✓
    ⊞ CVP cg::rd...          100.00%    100   100.00...  ✓
    ⊞ CVP cg::re...          100.00%    100   100.00...  ✓
    ⊞ CVP cg::ful...         100.00%    100   100.00...  ✓
    ⊞ CVP cg::e...           100.00%    100   100.00...  ✓
    ⊞ CVP cg::al...          100.00%    100   100.00...  ✓
    ⊞ CVP cg::al...          100.00%    100   100.00...  ✓
    ⊞ CVP cg::ov...          100.00%    100   100.00...  ✓
    ⊞ CVP cg::un...          100.00%    100   100.00...  ✓
    ⊞ CVP cg::wr...          100.00%    100   100.00...  ✓
    ⊞ CROSS cg:...           100.00%    100   100.00...  ✓
    ⊞ CROSS cg:...           100.00%    100   100.00...  ✓
    ⊞ CROSS cg:...           100.00%    100   100.00...  ✓
    ⊞ CROSS cg:...           100.00%    100   100.00...  ✓
    ⊞ CROSS cg:...           100.00%    100   100.00...  ✓
    ⊞ CROSS cg:...           100.00%    100   100.00...  ✓
    ⊞ CROSS cg:...           100.00%    100   100.00...  ✓
```



```
/FIFO_top/DUT/cover__threshold     SVA  ✓  Off  228  1  Unli...  1  100%  ✓
/FIFO_top/DUT/cover__R_wrapping    SVA  ✓  Off    6  1  Unli...  1  100%  ✓
/FIFO_top/DUT/cover__W_wrapping    SVA  ✓  Off   10  1  Unli...  1  100%  ✓
/FIFO_top/DUT/cover__almostempty   SVA  ✓  Off   17  1  Unli...  1  100%  ✓
/FIFO_top/DUT/cover__almostfull    SVA  ✓  Off   56  1  Unli...  1  100%  ✓
/FIFO_top/DUT/cover__full          SVA  ✓  Off   85  1  Unli...  1  100%  ✓
/FIFO_top/DUT/cover__empty         SVA  ✓  Off    8  1  Unli...  1  100%  ✓
/FIFO_top/DUT/cover__underflow     SVA  ✓  Off    4  1  Unli...  1  100%  ✓
/FIFO_top/DUT/cover__overflow      SVA  ✓  Off   58  1  Unli...  1  100%  ✓
/FIFO_top/DUT/cover__wr_ack        SVA  ✓  Off   98  1  Unli...  1  100%  ✓
```