FIFO Verification Project

➩ ***Verification plan:***

Label	Description	Stimulus Generation	Functional Coverage (Later)	Functionality Check
FIFO_1	When the reset is asserted, all outputs should be low except empty flag is high	Directed at the start of the simulation, then randomized with a constraint that make reset is off at most of simulation time	-	using immediate assertion to check the functionality of asynchronous reset , also using golden model
FIFO_2	checking when reset is disabled and write enable is high and full flag is low , write operation should be occured and write acknowldege rises , then write pointer is incremented	Randomized with constraint that write enable is high at 70% of simulation time and low at 30% of simulation time by default	cover the cases at which write enable is high and low and cover all cases between write enable , read enable , and write_ack ignoring the case at which write_en is low and write_ack is high (as write_ack is high only when write_en is high)	using concurrent assertion (wr_ack_ass property and wr_ptr_ass property) to check the functionality of write operation and also using golden model
FIFO_3	checking when reset is disabled and write enable is low or full flag is high , write operation shouldn't be occured and write acknowldege will be low , then write pointer is remained constant	Randomized with constraint that write enable is high at 70% of simulation time and low at 30% of simulation time by default	cover the cases at which write enable is high and low	using concurrent assertion (wr_ack_ass property and wr_ptr_ass property) to check that write acknowledge is low and write pointer isn't incremented as write operation isn't occurred and also using golden model
FIFO_4	checking overflow flag that when reset is disabled , if full flag is asserted and write enable is also asserted , then overflow flag should be high	Randomized	cover the cases by cross covering between write enable , read enable , and overflow flag ignoring the case at which write_en is low and overflow is high as it isn't important (as overflow may occur only when write_en is high)	using concurrent assertion (overflow_ass property) to check that overflow is high when both full flag and write enable are high , and also using golden model
FIFO_5	checking full flag that when reset is disabled , if count reached the FIFO depth (8) , then full flag should be high and no write operation can be occurred until read operation is occurred	Randomized	cover the cases by cross covering between write enable , read enable , and full flag ignoring the case at which read_en is high and full is high as it isn't important (as full flag may occur only when write_en is high)	using immediate assertion to check that full is high when count reached FIFO depth(8) , and also using golden model
FIFO_6	checking almostfull flag that when reset is disabled , if count reached the FIFO depth-1 (7) , then almostfull flag should be high and only one write operation can be occurred	Randomized	cover the cases by cross covering between write enable , read enable , and almostfull flag	using immediate assertion to check that almostfull is high when count reached FIFO depth-1(7) , and also using golden model
FIFO_7	checking when reset is disabled and read enable is high and empty flag is low , read operation should be occured and dataout takes the read value , then read pointer is incremented	Randomized with constraint that write enable is high at 30% of simulation time and low at 70% of simulation time by default	cover the cases at which read enable is high and low	using concurrent assertion (rd_ptr_ass property)to check the functionality of read operation and also using golden model
FIFO_8	checking underflow flag that when reset is disabled , if empty flag is asserted and read enable is also asserted , then underflow flag should be high	Randomized	cover the cases by cross covering between write enable , read enable , and underflow flag ignoring the case at which read_en is low and underflow is high as it isn't important (as underflow may occur only when read_en is high)	using concurrent assertion (underflow_ass property) to check that underflow is high when both empty flag and read enable are high , and also using golden model
FIFO_9	checking empty flag that when reset is disabled , if count reached zero , then empty flag should be high and no read operation can be occurred until write operation is occurred	Randomized	cover the cases by cross covering between write enable , read enable , and empty flag	using immediate assertion to check that empty is high when count reached zero , and also using golden model

FIFO_10	checking almostempty flag that when reset is disabled , if count reached zero, then almostempty flag should be high and only one read operation can be occurred	Randomized	cover the cases by cross covering between write enable , read enable , and almostempty flag	using immediate assertion to check that almostempty is high when count = 1 , and also using golden model
FIFO_11	check that when reset is disabled ,the count is incremented when write opertaion is occured	Randomization	-	using concurrent assertion (count_inc_ass property) to check that count is incremented when write opertaion is occured
FIFO_12	check that when reset is disabled ,the count is decremented when read opertaion is occured	Randomization	-	using concurrent assertion (count_dec_ass property) to check that count is decremented when read opertaion is occured
FIFO_13	check that when reset is disabled ,the count remains constant when write opertaion and read operation are occured at the same time	Randomization	-	using concurrent assertion (count_const_ass property) to check that count is constant when write opertaion and read operation are occured at the same time

⇨ ***Design before debugging:***

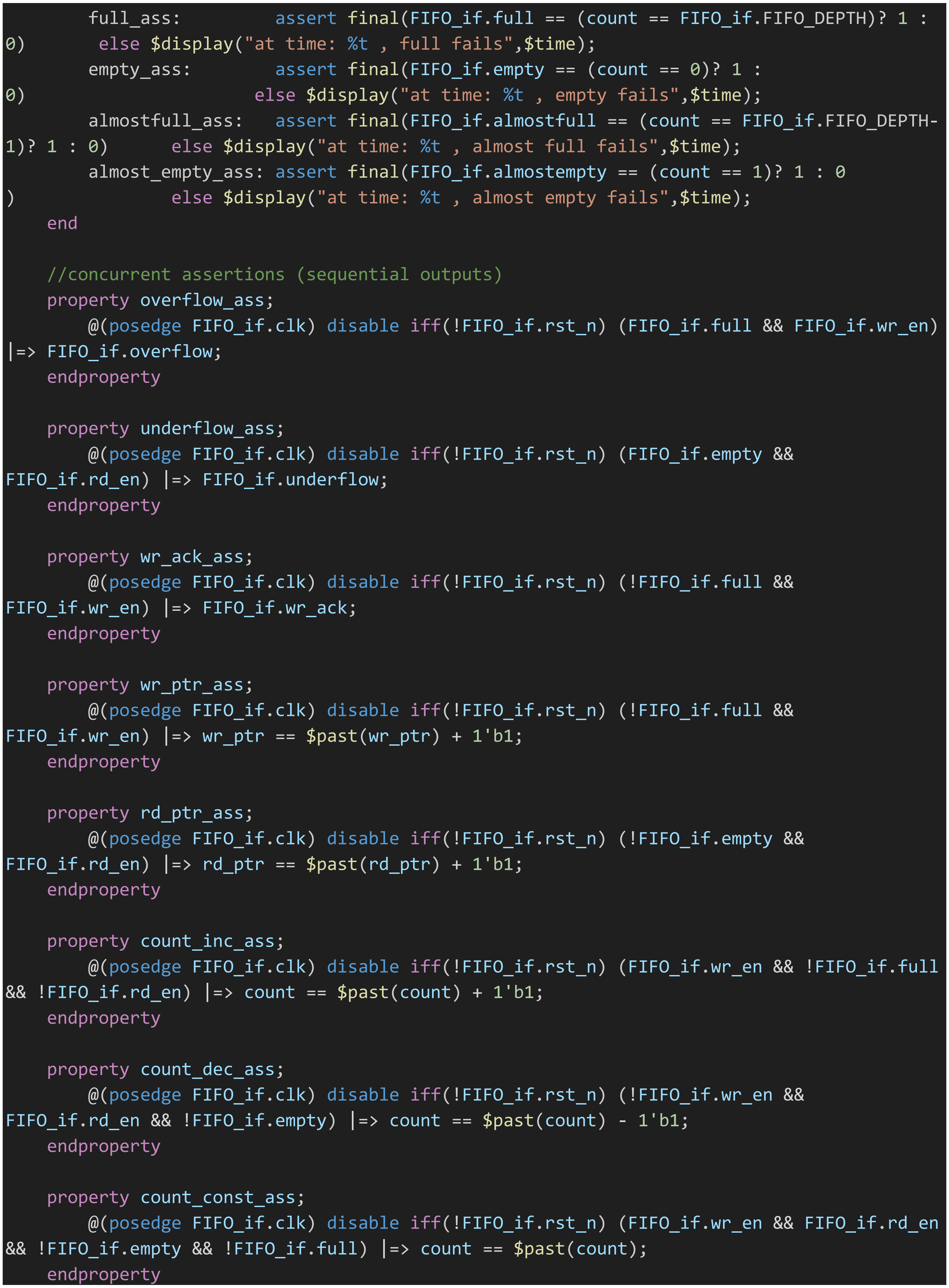
```
module FIFO(data_in, wr_en, rd_en, clk, rst_n, full, empty, almostfull, almostempty,
wr_ack, overflow, underflow, data_out);
parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;
input [FIFO_WIDTH-1:0] data_in;
input clk, rst_n, wr_en, rd_en;
output reg [FIFO_WIDTH-1:0] data_out;
output reg wr_ack, overflow;
output full, empty, almostfull, almostempty, 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;
    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;
        if (full & wr_en)
            overflow <= 1;
        else
            overflow <= 0;
    end
end
```

```verilog
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

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

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;

endmodule
```

➩ ***<u>Design after debugging:</u>***

```
module FIFO(FIFO_interface.DUT FIFO_if);

    localparam max_fifo_addr = $clog2(FIFO_if.FIFO_DEPTH);

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

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

    // write operation
    always @(posedge FIFO_if.clk or negedge FIFO_if.rst_n) begin
        if (!FIFO_if.rst_n) begin
            wr_ptr <= 0;
            FIFO_if.overflow <= 0;  //fix: overflow signal should be zero at reset
            FIFO_if.wr_ack <= 0;    //fix: write_ack signal should be zero at reset
        end
        else if (FIFO_if.wr_en && count < FIFO_if.FIFO_DEPTH) begin
            mem[wr_ptr] <= FIFO_if.data_in;
            FIFO_if.wr_ack <= 1;
            wr_ptr <= wr_ptr + 1;
            FIFO_if.overflow <= 0;  //fix: due to FIFO is not full , so overflow should be
zero
        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

    // read operation
    always @(posedge FIFO_if.clk or negedge FIFO_if.rst_n) begin
        if (!FIFO_if.rst_n) begin
            rd_ptr <= 0;
            FIFO_if.underflow <= 0;   //fix: underflow signal should be zero at reset
            FIFO_if.data_out <= 0;    //fix: dataout signal should be zero at reset
        end
        else if (FIFO_if.rd_en && count != 0) begin
            FIFO_if.data_out <= mem[rd_ptr];
            rd_ptr <= rd_ptr + 1;
            FIFO_if.underflow <= 0;    //fix: due to FIFO is not empty , so underflow
should be zero
        end

        else begin   //fix : underflow output is sequential output not combinational
            if (FIFO_if.rd_en && FIFO_if.empty) begin
```

```
                FIFO_if.underflow <= 1;       //fix
            end
            else begin
                FIFO_if.underflow <= 0;        //fix
            end
        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'b11) && FIFO_if.full) begin
                    count <= count-1;  //fix: when both wr_en and rd_en are high , and
full=1 , only read operation will occur
            end
            else if (({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b11) && FIFO_if.empty)
begin  //fix
                    count <= count+1; //fix: when both wr_en and rd_en are high , and
empty=1 , only write operation will occur
            end
            else if (({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b11) && !FIFO_if.full &&
!FIFO_if.empty) begin  //fix
                    count <= count; //fix: when both wr_en and rd_en are high , and both
empty=0 and full=0 , both operations (read,write) will occur
            end
            else 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;
        end
    end

    assign FIFO_if.full = (count == FIFO_if.FIFO_DEPTH)? 1 : 0;
    assign FIFO_if.empty = (count == 0)? 1 : 0;
    assign FIFO_if.almostfull = (count == FIFO_if.FIFO_DEPTH-1)? 1 : 0; //fix : almostfull
signal is high when count=FIFO_DEPTH-1 not FIFO_DEPTH-2
    assign FIFO_if.almostempty = (count == 1)? 1 : 0;


    //FIFO_Assertions
    `ifdef SIM

    //immediate assertions (combinational outputs)
    always_comb begin
        if(!FIFO_if.rst_n) begin
            reset_ass: assert final((!count) && (!rd_ptr) && (!wr_ptr))
            else $display("at time: %t , reset fails",$time);
        end
```



```
        full_ass:         assert final(FIFO_if.full == (count == FIFO_if.FIFO_DEPTH)? 1 :
0)       else $display("at time: %t , full fails",$time);
        empty_ass:        assert final(FIFO_if.empty == (count == 0)? 1 :
0)                      else $display("at time: %t , empty fails",$time);
        almostfull_ass:   assert final(FIFO_if.almostfull == (count == FIFO_if.FIFO_DEPTH-
1)? 1 : 0)      else $display("at time: %t , almost full fails",$time);
        almost_empty_ass: assert final(FIFO_if.almostempty == (count == 1)? 1 : 0
)               else $display("at time: %t , almost empty fails",$time);
    end

    //concurrent assertions (sequential outputs)
    property overflow_ass;
        @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.full && FIFO_if.wr_en)
|=> FIFO_if.overflow;
    endproperty

    property underflow_ass;
        @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.empty &&
FIFO_if.rd_en) |=> FIFO_if.underflow;
    endproperty

    property wr_ack_ass;
        @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (!FIFO_if.full &&
FIFO_if.wr_en) |=> FIFO_if.wr_ack;
    endproperty

    property wr_ptr_ass;
        @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (!FIFO_if.full &&
FIFO_if.wr_en) |=> wr_ptr == $past(wr_ptr) + 1'b1;
    endproperty

    property rd_ptr_ass;
        @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (!FIFO_if.empty &&
FIFO_if.rd_en) |=> rd_ptr == $past(rd_ptr) + 1'b1;
    endproperty

    property count_inc_ass;
        @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.wr_en && !FIFO_if.full
&& !FIFO_if.rd_en) |=> count == $past(count) + 1'b1;
    endproperty

    property count_dec_ass;
        @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (!FIFO_if.wr_en &&
FIFO_if.rd_en && !FIFO_if.empty) |=> count == $past(count) - 1'b1;
    endproperty

    property count_const_ass;
        @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.wr_en && FIFO_if.rd_en
&& !FIFO_if.empty && !FIFO_if.full) |=> count == $past(count);
    endproperty
    overflow_assert: assert property(overflow_ass) else $display("at time %t : overflow
Fails",$time);
    underflow_assert: assert property(underflow_ass) else $display("at time %t : underflow
Fails",$time);
    wr_ack_assert: assert property(wr_ack_ass) else $display("at time %t : write ack
Fails",$time);
    wr_ptr_assert: assert property(wr_ptr_ass) else $display("at time %t : write pointer
Fails",$time);
    rd_ptr_assert: assert property(rd_ptr_ass) else $display("at time %t : read pointer
Fails",$time);
    count_inc_assert: assert property(count_inc_ass) else $display("at time %t : counter
increment Fails",$time);
    count_dec_assert: assert property(count_dec_ass) else $display("at time %t : counter
decrement Fails",$time);
    count_const_assert: assert property(count_const_ass) else $display("at time %t :
counter const Fails",$time);

    overflow_cover: cover property (overflow_ass);
    underflow_cover: cover property (underflow_ass);
    wr_ack_cover: cover property (wr_ack_ass);
    wr_ptr_cover: cover property (wr_ptr_ass);
    rd_ptr_cover: cover property (rd_ptr_ass);
    count_inc_cover: cover property (count_inc_ass);
    count_dec_cover: cover property (count_dec_ass);
    count_const_cover: cover property (count_const_ass);
    `endif
endmodule
```

➪ ***Interface:***

```
interface FIFO_interface(clk);
    input clk;
    parameter FIFO_WIDTH = 16;
    parameter FIFO_DEPTH = 8;
    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;
    modport DUT (input data_in, wr_en, rd_en, clk, rst_n , output full, empty, almostfull,
almostempty, wr_ack, overflow, underflow, data_out);

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

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

➪ ***<u>Transaction package code:</u>***

```
package FIFO_transaction_pkg;
    class FIFO_transaction;
        parameter FIFO_WIDTH = 16;
        parameter FIFO_DEPTH = 8;
        rand logic [FIFO_WIDTH-1:0] data_in;
        rand logic rst_n, wr_en, rd_en;
        logic [FIFO_WIDTH-1:0] data_out;
        logic wr_ack, overflow;
        logic full, empty, almostfull, almostempty, underflow;
        int RD_EN_ON_DIST;
        int WR_EN_ON_DIST;

        constraint reset_c {
            rst_n dist {1:=96 , 0:=4};
        }

        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};
        }

        function new(input int WR_EN_ON_DIST = 70 , RD_EN_ON_DIST = 30);
            this.RD_EN_ON_DIST = RD_EN_ON_DIST;
            this.WR_EN_ON_DIST = WR_EN_ON_DIST;
        endfunction
    endclass
endpackage
```

➪ ***<u>Shared package:</u>***

```
package shared_pkg;
    int error_count = 0;
    int correct_count = 0;
    bit test_finished = 0;
endpackage
```

⇨ ***Testbench code:***

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

module FIFO_tb (FIFO_interface.TEST FIFO_if);
    parameter FIFO_WIDTH = FIFO_if.FIFO_DEPTH;
    parameter FIFO_DEPTH = FIFO_if.FIFO_WIDTH;

    // create object
    FIFO_transaction tr_obj = new();


    initial begin
        // initialize design
        FIFO_if.rst_n = 0;
        FIFO_if.wr_en = 1;
        FIFO_if.rd_en = 0;
        FIFO_if.data_in = 5;
        @(negedge FIFO_if.clk);
        #0;

        for (int i = 0;i<1000 ;i++ ) begin
            assert(tr_obj.randomize());
            FIFO_if.rst_n = tr_obj.rst_n;
            FIFO_if.wr_en = tr_obj.wr_en;
            FIFO_if.rd_en = tr_obj.rd_en;
            FIFO_if.data_in = tr_obj.data_in;
            @(negedge FIFO_if.clk);
        end

        // end test
        test_finished = 1;
        #1;
    end
endmodule
```

⇨ ***Coverage package:***

```
package FIFO_coverage_pkg;
    import FIFO_transaction_pkg::*;
    class FIFO_coverage;
        FIFO_transaction F_cvg_txn = new();

        covergroup write_read_cover;
            write_enable_cvg: coverpoint F_cvg_txn.wr_en;          //coverpoint for
write_en signal
            read_enable_cvg : coverpoint F_cvg_txn.rd_en;          //coverpoint for
read_en signal
            full_cvg:         coverpoint F_cvg_txn.full;           //coverpoint for full
flag output
            empty_cvg:        coverpoint F_cvg_txn.empty;          //coverpoint for empty
flag output
            almost_full_cvg:  coverpoint F_cvg_txn.almostfull;     //coverpoint for
almostfull flag output
            almost_empty_cvg: coverpoint F_cvg_txn.almostempty;    //coverpoint for
almostempty flag output
            write_ack_cvg:    coverpoint F_cvg_txn.wr_ack;         //coverpoint for
write_ack flag output
            overflow_cvg:     coverpoint F_cvg_txn.overflow;       //coverpoint for
overflow flag output
            underflow_cvg:    coverpoint F_cvg_txn.underflow;      //coverpoint for
underflow flag output

            write_read_full:        cross
write_enable_cvg,read_enable_cvg,full_cvg{                      // cross between wr_en ,
rd_en , full
                //not important for full output if write_en = 1 (as full=1 may only when
wr_en=1)
                ignore_bins full_read_en00 = binsof(read_enable_cvg) intersect {1} &&
binsof(full_cvg) intersect {1};
            }

            write_read_empty:       cross
write_enable_cvg,read_enable_cvg,empty_cvg;                // cross between wr_en , rd_en ,
empty
            write_read_almost_full: cross
write_enable_cvg,read_enable_cvg,almost_full_cvg;          // cross between wr_en , rd_en ,
almostfull
            write_read_almostempty: cross
write_enable_cvg,read_enable_cvg,almost_empty_cvg;         // cross between wr_en , rd_en ,
almostempty

            write_read_wr_ack:      cross
write_enable_cvg,read_enable_cvg,write_ack_cvg{            // cross between wr_en , rd_en
, wr_ack
```

```
                //not important for wr_ack output if write_en = 0 (as wr_ack=1 only when
wr_en=1)
                ignore_bins wr_ack_wr_en00 = binsof(write_enable_cvg) intersect {0} &&
binsof(write_ack_cvg) intersect {1};
            }

            write_read_overflow:    cross
write_enable_cvg,read_enable_cvg,overflow_cvg{             // cross between wr_en , rd_en
, overflow
                //not important for overflow output if write_en = 0 (as overflow occurs
only when wr_en=1)
                ignore_bins write_overflow00 = binsof(write_enable_cvg) intersect {0} &&
binsof(overflow_cvg) intersect {1};
            }

            write_read_underflow:   cross
write_enable_cvg,read_enable_cvg,underflow_cvg{            // cross between wr_en , rd_en
, underflow
                //not important for underflow output if read_en = 0 (as underflow occurs
only when rd_en=1)
                ignore_bins read_underflow00 = binsof(read_enable_cvg) intersect {0} &&
binsof(underflow_cvg) intersect {1};
            }
        endgroup

        function void sample_data(input FIFO_transaction F_txn);
            this.F_cvg_txn = F_txn;
            this.write_read_cover.sample();    //sampling the covergroup
        endfunction

        function new();
            write_read_cover = new();
        endfunction
    endclass
endpackage
```

⇨ ***Scoreboard package:***

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

    class FIFO_scoreboard;
        //parameters
        parameter FIFO_WIDTH = 16;
        parameter FIFO_DEPTH = 8;
        localparam max_fifo_addr = $clog2(FIFO_DEPTH);

        //internal signal
        logic [FIFO_WIDTH-1:0] mem [FIFO_DEPTH-1:0];
        logic [max_fifo_addr-1:0] wr_ptr, rd_ptr;
        logic [max_fifo_addr:0] count;

        // refernce output
        logic[FIFO_WIDTH-1:0] data_out_ref;
        logic wr_ack_ref, overflow_ref;
        logic full_ref, empty_ref, almostfull_ref, almostempty_ref, underflow_ref;

        function void check_data(FIFO_transaction checked_data_obj);
            reference_model(checked_data_obj);
            if (data_out_ref != checked_data_obj.data_out || wr_ack_ref != 
checked_data_obj.wr_ack || overflow_ref != checked_data_obj.overflow ||
            full_ref != checked_data_obj.full || empty_ref != checked_data_obj.empty || 
almostfull_ref != checked_data_obj.almostfull ||
            almostempty_ref != checked_data_obj.almostempty || underflow_ref != 
checked_data_obj.underflow) begin

                $display("at time: %0t Error, Incorrect FIFO",$time);
                error_count++;
            end
            else begin
                correct_count++;
            end
        endfunction

        function void reference_model(FIFO_transaction golden_model_obj);
            // write operation
            if (!golden_model_obj.rst_n) begin
                wr_ptr = 0;
                overflow_ref = 0;
                wr_ack_ref = 0;
            end
            else if (golden_model_obj.wr_en && !full_ref) begin
                mem[wr_ptr] = golden_model_obj.data_in;
                wr_ack_ref = 1;
                wr_ptr = wr_ptr + 1;
                overflow_ref = 0;
            end
            else begin
                wr_ack_ref = 0;
                if (full_ref && golden_model_obj.wr_en) begin
                    overflow_ref = 1;
                end
                else begin
                    overflow_ref = 0;
                end
            end

            //read operation
            if (!golden_model_obj.rst_n) begin
                rd_ptr = 0;
                underflow_ref = 0;
                data_out_ref = 0;
            end
            else if (golden_model_obj.rd_en && !empty_ref) begin
                data_out_ref = mem[rd_ptr];
                rd_ptr = rd_ptr + 1;
                underflow_ref = 0;
            end

            else begin
                if (golden_model_obj.rd_en && empty_ref) begin
                    underflow_ref = 1;
                end
                else begin
                    underflow_ref = 0;
                end
            end

            // count calculation
        if (!golden_model_obj.rst_n) begin
            count = 0;
        end
        else begin
            if (({golden_model_obj.wr_en, golden_model_obj.rd_en} == 2'b11) && full_ref)
begin //fix
                    count = count-1;
            end
            else if (({golden_model_obj.wr_en, golden_model_obj.rd_en} == 2'b11) &&
empty_ref) begin  //fix
                    count = count+1;
            end
            else if (({golden_model_obj.wr_en, golden_model_obj.rd_en} == 2'b11) &&
!full_ref && !empty_ref) begin  //fix
                    count = count;
            end
            else if ( ({golden_model_obj.wr_en,golden_model_obj.rd_en} == 2'b10) &&
!full_ref)
                count = count + 1;
            else if ( ({golden_model_obj.wr_en, golden_model_obj.rd_en} == 2'b01) &&
!empty_ref)
                count = count - 1;
        end
            // assign combinational signals (full,empty,almostfull,almostempty)
            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;

        endfunction
    endclass
endpackage
```

➪ ***Monitor:***

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

module FIFO_monitor (FIFO_interface.MONITOR FIFO_if);
    FIFO_transaction tr_monitor = new();
    FIFO_coverage cov_monitor = new();
    FIFO_scoreboard score_monitor = new();

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

            fork
                begin
                    cov_monitor.sample_data(tr_monitor);
                end
                begin
                    @(posedge FIFO_if.clk);
                    score_monitor.check_data(tr_monitor);
                end
            join

            //ending simulation
            if (test_finished == 1) begin
                $display("****************************************************************
");
                $display("****************************************************************
");
                $display("************************Test
Summary****************************");
                $display("the design passed by %0d correct outputs and %0d errors
",correct_count,error_count);
                $display("****************************************************************
");
                $display("****************************************************************
");
                $stop;
            end
        end
    end
endmodule
```

⇨ ***Top testbench:***

```
module FIFO_top ();
    bit clk;
    initial begin
        clk = 1;
        forever begin
           #1 clk = ~clk;

        end
    end

    FIFO_interface FIFO_if(clk);
    FIFO_tb tb(FIFO_if);
    FIFO DUT(FIFO_if);
    FIFO_monitor MONITOR(FIFO_if);
endmodule
```

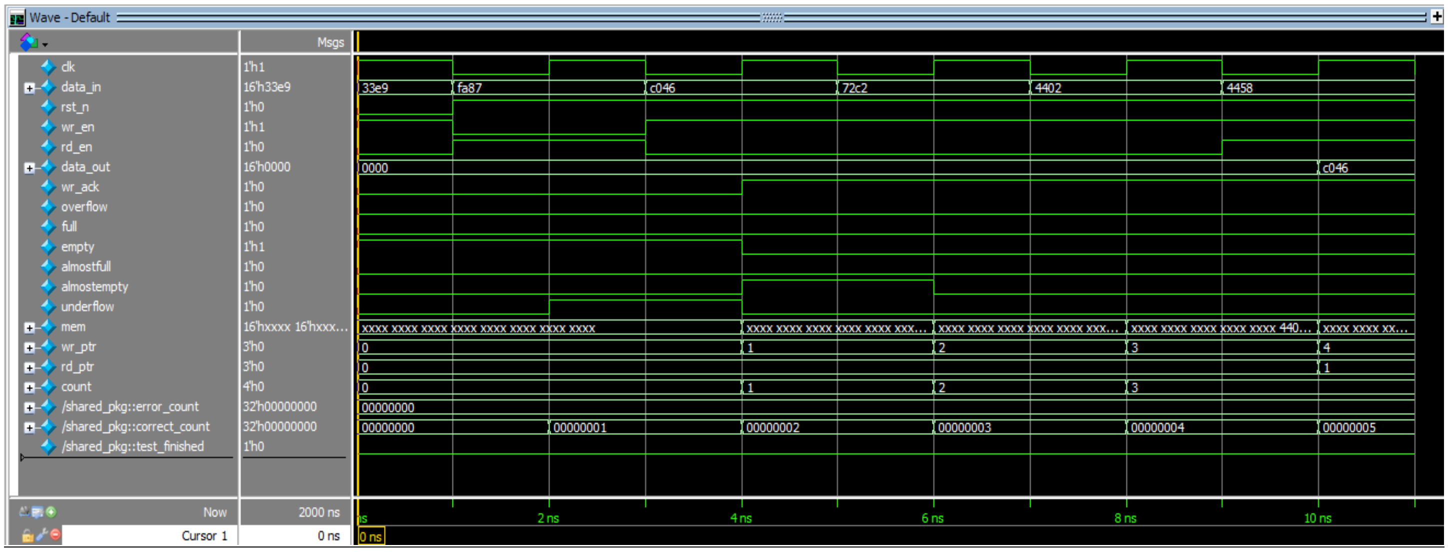
Do file:

```
vlib work
vlog +define+SIM +cover -covercells FIFO.sv FIFO_tb.sv FIFO_transaction_pkg.sv
FIFO_coverage_pkg.sv FIFO_scoreboard.sv FIFO_monitor.sv FIFO_interface.sv shared_pkg.sv
FIFO_top.sv
vsim -voptargs=+acc work.FIFO_top -cover
add wave *
add wave -position insertpoint sim:/FIFO_top/FIFO_if/*
add wave -position insertpoint  \
sim:/FIFO_top/DUT/mem \
sim:/FIFO_top/DUT/wr_ptr \
sim:/FIFO_top/DUT/rd_ptr \
sim:/FIFO_top/DUT/count
add wave -position insertpoint  \
sim:/shared_pkg::error_count \
sim:/shared_pkg::correct_count \
sim:/shared_pkg::test_finished
coverage save FIFO_tb.ucdb -onexit
run -all
```

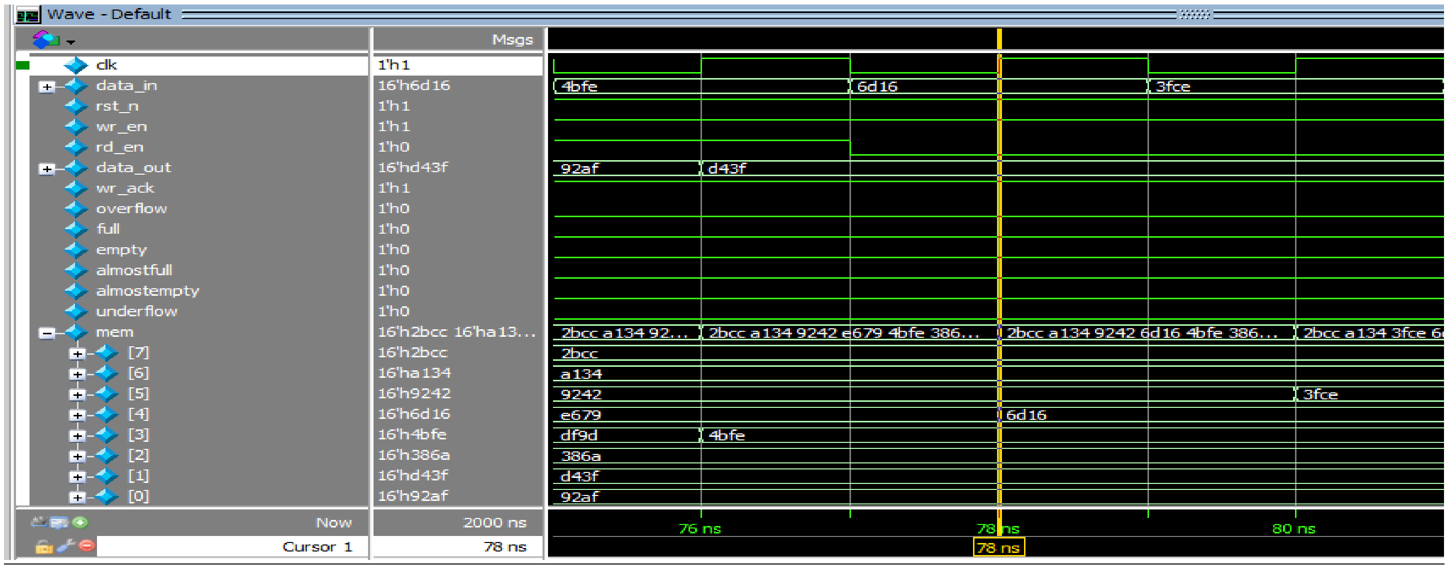
Detected bugs:

1) *Almost full should be high when count = FIFO_DEPTH – 1 not count = FIFO_DEPTH – 2*
2) *Each of overflow, underflow, wr_ack, dataout signals should be zero at reset*
3) *We should give overflow signal zero when FIFO is not full and write operation is done successfully*
4) *We should give underflow signal zero when FIFO is not empty and read operation is done successfully*
5) *Underflow output is sequential output not combinational. so, it should be get from always block*
6) *At the always block of count internal signal we should take into consideration some uncovered cases:*
    a) *When wr_en and rd_en are high together and full = 1 so, count should be decremented.*
    b) *When wr_en and rd_en are high together and empty = 1 so, count should be incremented.*
    c) *When wr_en and rd_en are high together and both full and empty flags are low so, count should remain constant.*

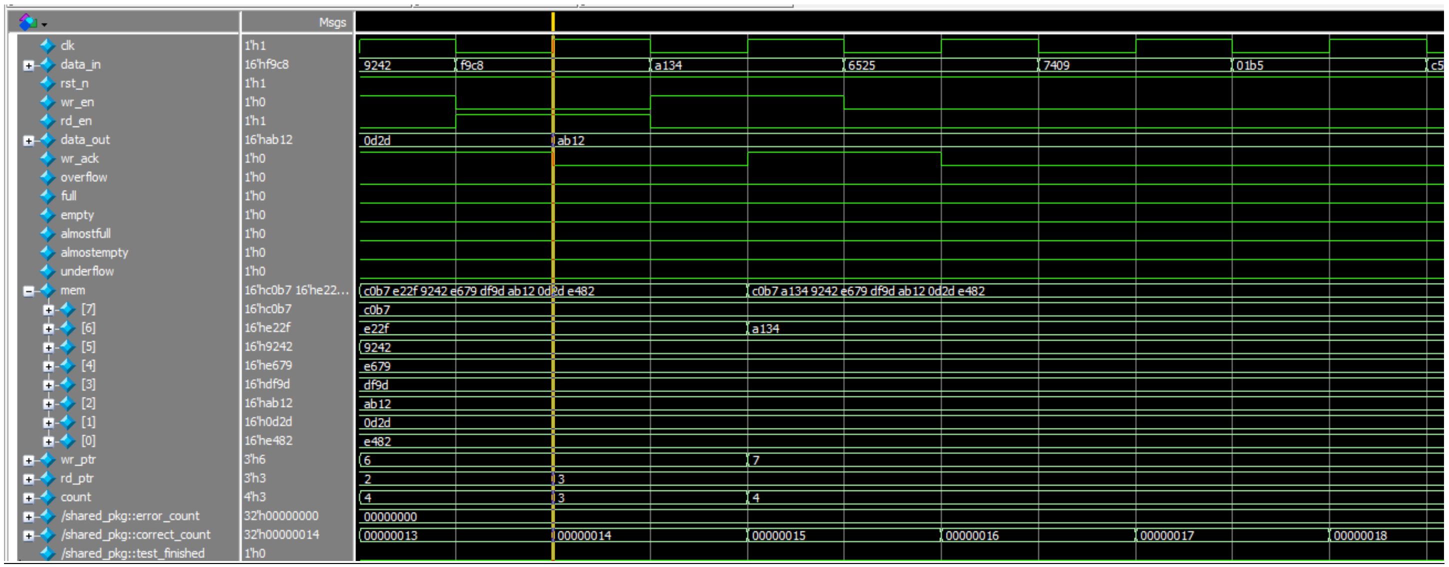
⇨ ***Questasim snippets:***



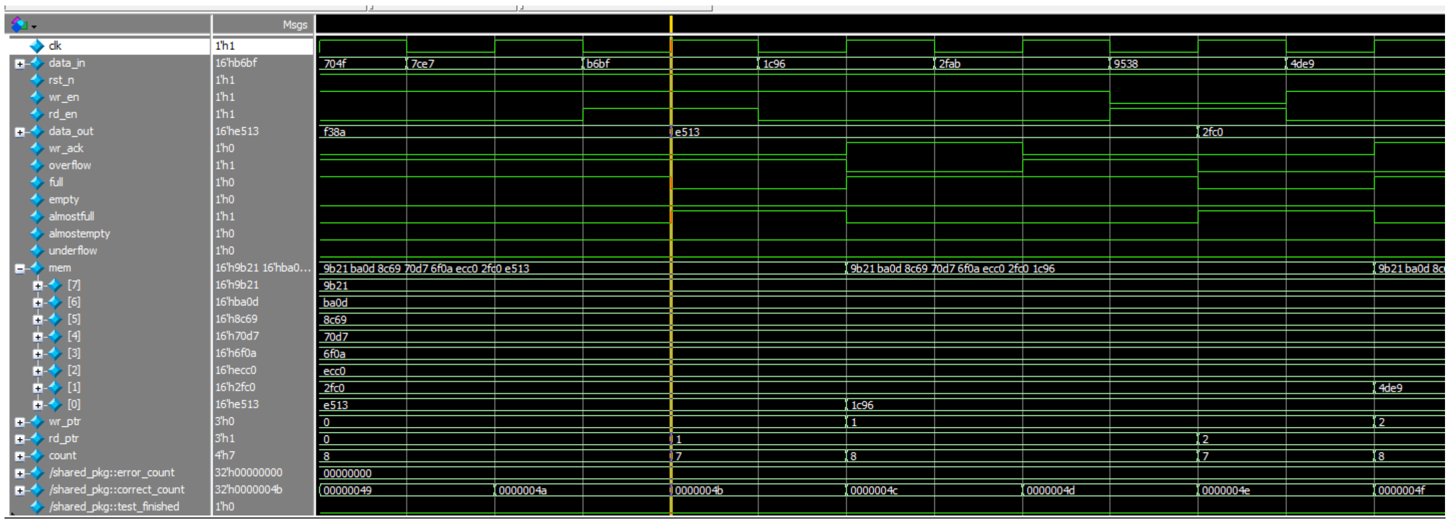
```
Wave - Default
                              Msgs
dk                            1'h1
data_in                       16'h33e9
rst_n                         1'h0
wr_en                         1'h1
rd_en                         1'h0
data_out                      16'h0000
wr_ack                        1'h0
overflow                      1'h0
full                          1'h0
empty                         1'h1
almostfull                    1'h0
almostempty                   1'h0
underflow                     1'h0
mem                           16'hxxxx 16'hxxx...
wr_ptr                        3'h0
rd_ptr                        3'h0
count                         4'h0
/shared_pkg::error_count      32'h00000000
/shared_pkg::correct_count    32'h00000000
/shared_pkg::test_finished    1'h0
Now                           2000 ns
Cursor 1                      0 ns
```



```
Wave - Default
                    Msgs
dk                  1'h1
data_in             16'h6d16
rst_n               1'h1
wr_en               1'h1
rd_en               1'h0
data_out            16'hd43f
wr_ack              1'h1
overflow            1'h0
full                1'h0
empty               1'h0
almostfull          1'h0
almostempty         1'h0
underflow           1'h0
mem                 16'h2bcc 16'ha13...
  [7]               16'h2bcc
  [6]               16'ha134
  [5]               16'h9242
  [4]               16'h6d16
  [3]               16'h4bfe
  [2]               16'h386a
  [1]               16'hd43f
  [0]               16'h92af
Now                 2000 ns
Cursor 1            78 ns
```



```
                              Msgs
dk                            1'h1
data_in                       16'hf9c8
rst_n                         1'h1
wr_en                         1'h0
rd_en                         1'h1
data_out                      16'hab12
wr_ack                        1'h0
overflow                      1'h0
full                          1'h0
empty                         1'h0
almostfull                    1'h0
almostempty                   1'h0
underflow                     1'h0
mem                           16'hc0b7 16'he22...
  [7]                         16'hc0b7
  [6]                         16'he22f
  [5]                         16'h9242
  [4]                         16'he679
  [3]                         16'hdf9d
  [2]                         16'hab12
  [1]                         16'h0d2d
  [0]                         16'he482
wr_ptr                        3'h6
rd_ptr                        3'h3
count                         4'h3
/shared_pkg::error_count      32'h00000000
/shared_pkg::correct_count    32'h00000014
/shared_pkg::test_finished    1'h0
```

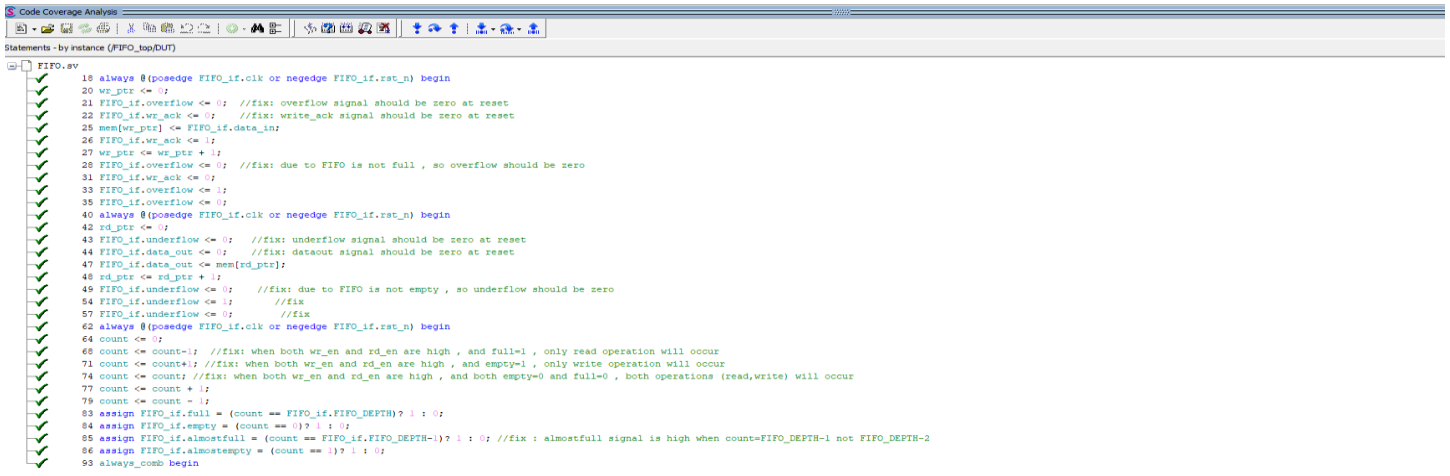


```
Msgs
clk                             1'h1
data_in                         16'hb6bf
rst_n                           1'h1
wr_en                           1'h1
rd_en                           1'h1
data_out                        16'he513
wr_ack                          1'h0
overflow                        1'h1
full                            1'h0
empty                           1'h0
almostfull                      1'h1
almostempty                     1'h0
underflow                       1'h0
mem                             16'h9b21 16'hba0...
  [7]                           16'h9b21
  [6]                           16'hba0d
  [5]                           16'h8c69
  [4]                           16'h70d7
  [3]                           16'h6f0a
  [2]                           16'hecc0
  [1]                           16'h2fc0
  [0]                           16'he513
wr_ptr                          3'h0
rd_ptr                          3'h1
count                           4'h7
/shared_pkg::error_count        32'h00000000
/shared_pkg::correct_count      32'h0000004b
/shared_pkg::test_finished      1'h0
```

```
# *****************************************************************
# *****************************************************************
# ************************Test Summary*****************************
# the design passed by 1000 correct outputs and 0 errors
# *****************************************************************
# *****************************************************************
# ** Note: $stop    : FIFO_monitor.sv(53)
#    Time: 2 us  Iteration: 1  Instance: /FIFO_top/MONITOR
# Break in Module FIFO_monitor at FIFO_monitor.sv line 53
```

⇨ ***Code coverage:***

1) ***Statement:***



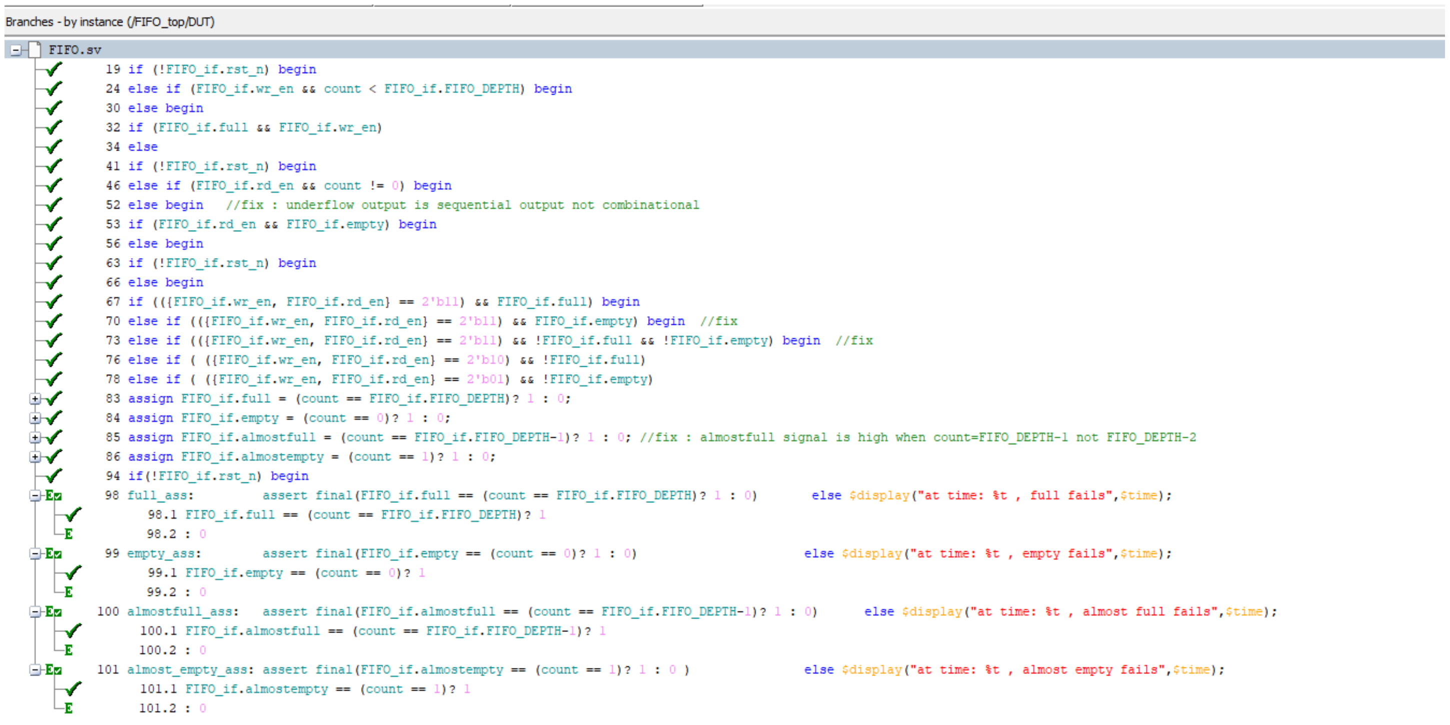
Code Coverage Analysis

Statements - by instance (/FIFO_top/DUT)

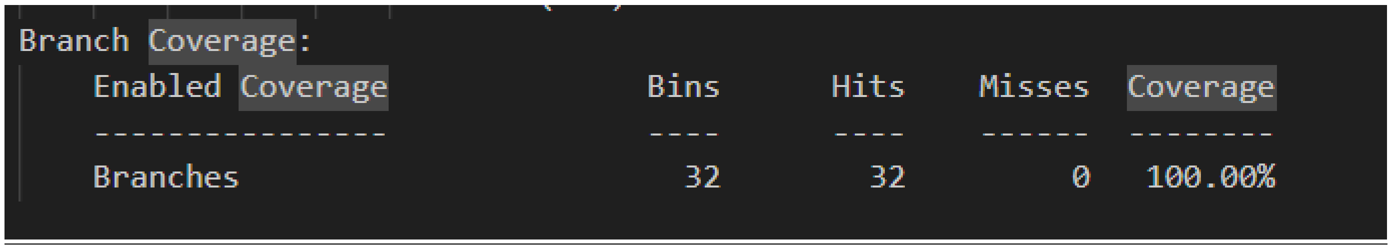
```
FIFO.sv
✓ 18 always @(posedge FIFO_if.clk or negedge FIFO_if.rst_n) begin
✓ 20 wr_ptr <= 0;
✓ 21 FIFO_if.overflow <= 0;  //fix: overflow signal should be zero at reset
✓ 22 FIFO_if.wr_ack <= 0;    //fix: write_ack signal should be zero at reset
✓ 25 mem[wr_ptr] <= FIFO_if.data_in;
✓ 26 FIFO_if.wr_ack <= 1;
✓ 27 wr_ptr <= wr_ptr + 1;
✓ 28 FIFO_if.overflow <= 0;  //fix: due to FIFO is not full , so overflow should be zero
✓ 31 FIFO_if.wr_ack <= 0;
✓ 33 FIFO_if.overflow <= 1;
✓ 35 FIFO_if.overflow <= 0;
✓ 40 always @(posedge FIFO_if.clk or negedge FIFO_if.rst_n) begin
✓ 42 rd_ptr <= 0;
✓ 43 FIFO_if.underflow <= 0;   //fix: underflow signal should be zero at reset
✓ 44 FIFO_if.data_out <= 0;    //fix: dataout signal should be zero at reset
✓ 47 FIFO_if.data_out <= mem[rd_ptr];
✓ 48 rd_ptr <= rd_ptr + 1;
✓ 49 FIFO_if.underflow <= 0;    //fix: due to FIFO is not empty , so underflow should be zero
✓ 54 FIFO_if.underflow <= 1;       //fix
✓ 57 FIFO_if.underflow <= 0;        //fix
✓ 62 always @(posedge FIFO_if.clk or negedge FIFO_if.rst_n) begin
✓ 64 count <= 0;
✓ 68 count <= count-1;  //fix: when both wr_en and rd_en are high , and full=1 , only read operation will occur
✓ 71 count <= count+1; //fix: when both wr_en and rd_en are high , and empty=1 , only write operation will occur
✓ 74 count <= count; //fix: when both wr_en and rd_en are high , and both empty=0 and full=0 , both operations (read,write) will occur
✓ 77 count <= count + 1;
✓ 79 count <= count - 1;
✓ 83 assign FIFO_if.full = (count == FIFO_if.FIFO_DEPTH)? 1 : 0;
✓ 84 assign FIFO_if.empty = (count == 0)? 1 : 0;
✓ 85 assign FIFO_if.almostfull = (count == FIFO_if.FIFO_DEPTH-1)? 1 : 0; //fix : almostfull signal is high when count=FIFO_DEPTH-1 not FIFO_DEPTH-2
✓ 86 assign FIFO_if.almostempty = (count == 1)? 1 : 0;
✓ 93 always_comb begin
```

```
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                      32        32         0   100.00%
```

2) ***Branch:***



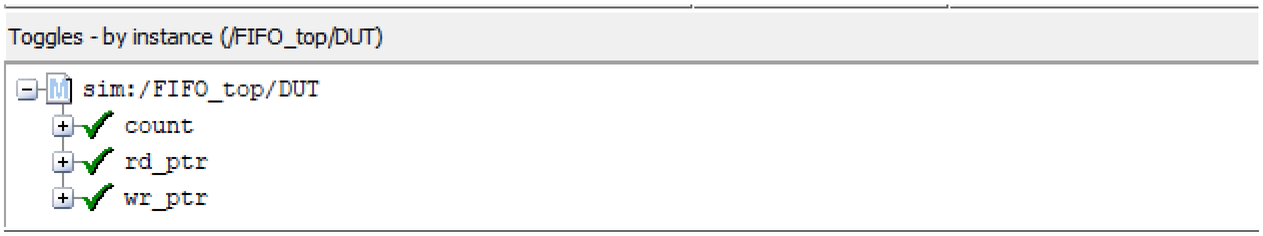
```
Branches - by instance (/FIFO_top/DUT)
FIFO.sv
     19 if (!FIFO_if.rst_n) begin
     24 else if (FIFO_if.wr_en && count < FIFO_if.FIFO_DEPTH) begin
     30 else begin
     32 if (FIFO_if.full && FIFO_if.wr_en)
     34 else
     41 if (!FIFO_if.rst_n) begin
     46 else if (FIFO_if.rd_en && count != 0) begin
     52 else begin   //fix : underflow output is sequential output not combinational
     53 if (FIFO_if.rd_en && FIFO_if.empty) begin
     56 else begin
     63 if (!FIFO_if.rst_n) begin
     66 else begin
     67 if (({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b11) && FIFO_if.full) begin
     70 else if (({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b11) && FIFO_if.empty) begin  //fix
     73 else if (({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b11) && !FIFO_if.full && !FIFO_if.empty) begin  //fix
     76 else if ( ({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b10) && !FIFO_if.full)
     78 else if ( ({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b01) && !FIFO_if.empty)
     83 assign FIFO_if.full = (count == FIFO_if.FIFO_DEPTH)? 1 : 0;
     84 assign FIFO_if.empty = (count == 0)? 1 : 0;
     85 assign FIFO_if.almostfull = (count == FIFO_if.FIFO_DEPTH-1)? 1 : 0; //fix : almostfull signal is high when count=FIFO_DEPTH-1 not FIFO_DEPTH-2
     86 assign FIFO_if.almostempty = (count == 1)? 1 : 0;
     94 if(!FIFO_if.rst_n) begin
     98 full_ass:        assert final(FIFO_if.full == (count == FIFO_if.FIFO_DEPTH)? 1 : 0)       else $display("at time: %t , full fails",$time);
        98.1 FIFO_if.full == (count == FIFO_if.FIFO_DEPTH)? 1
        98.2 : 0
     99 empty_ass:       assert final(FIFO_if.empty == (count == 0)? 1 : 0)                      else $display("at time: %t , empty fails",$time);
        99.1 FIFO_if.empty == (count == 0)? 1
        99.2 : 0
    100 almostfull_ass:  assert final(FIFO_if.almostfull == (count == FIFO_if.FIFO_DEPTH-1)? 1 : 0)      else $display("at time: %t , almost full fails",$time);
       100.1 FIFO_if.almostfull == (count == FIFO_if.FIFO_DEPTH-1)? 1
       100.2 : 0
    101 almost_empty_ass: assert final(FIFO_if.almostempty == (count == 1)? 1 : 0 )               else $display("at time: %t , almost empty fails",$time);
       101.1 FIFO_if.almostempty == (count == 1)? 1
       101.2 : 0
```



```
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        32        32         0   100.00%
```

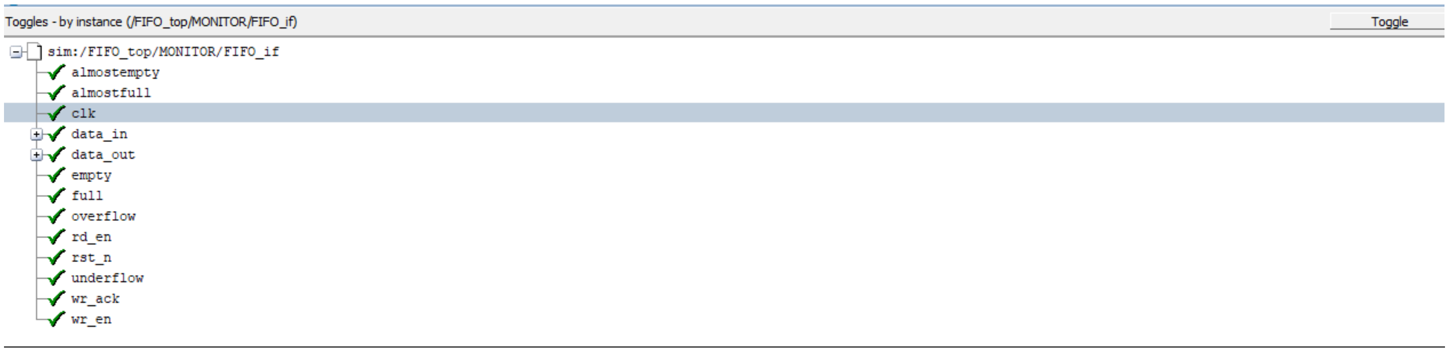
Note: I have excluded the branches which make the assertion returns zero, as it is meaningless for the assertion to return zero (I always wants the assertion to return true)

3) Toggle:



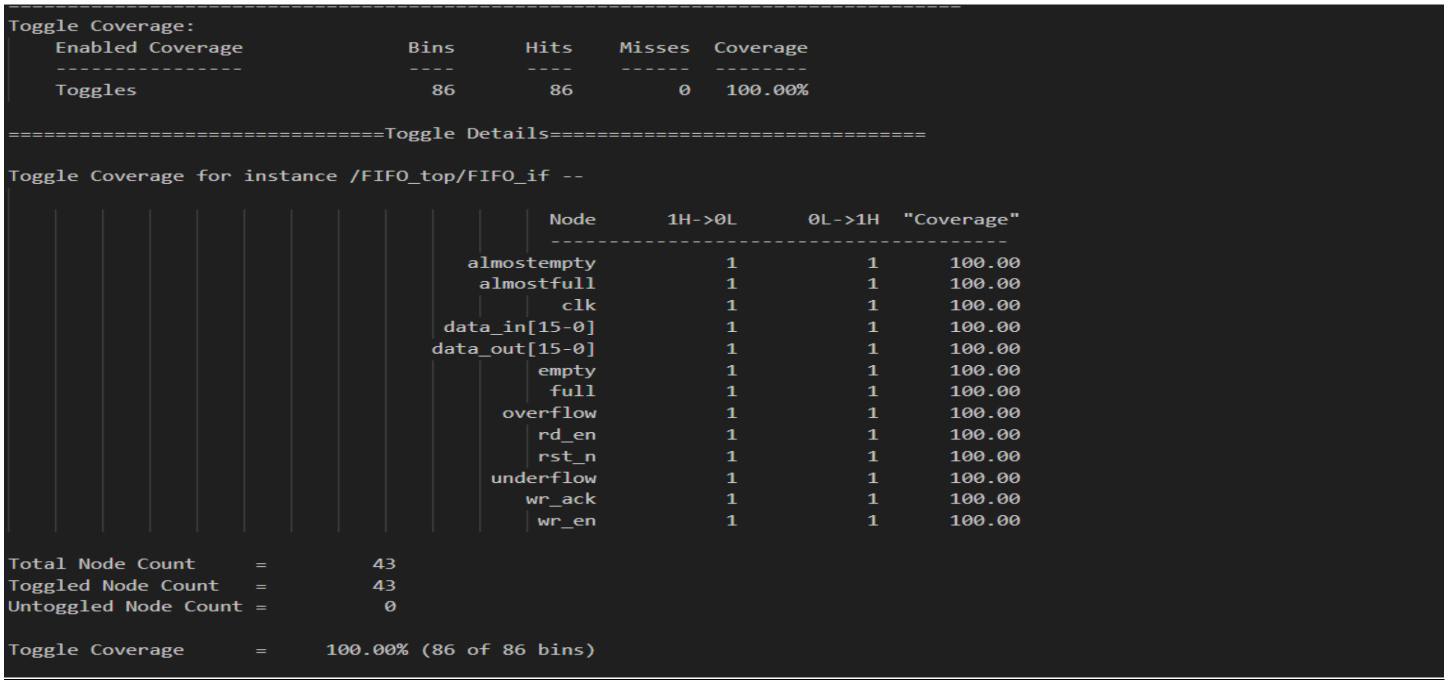
Toggles - by instance (/FIFO_top/DUT)

```
sim:/FIFO_top/DUT
  ✓ count
  ✓ rd_ptr
  ✓ wr_ptr
```



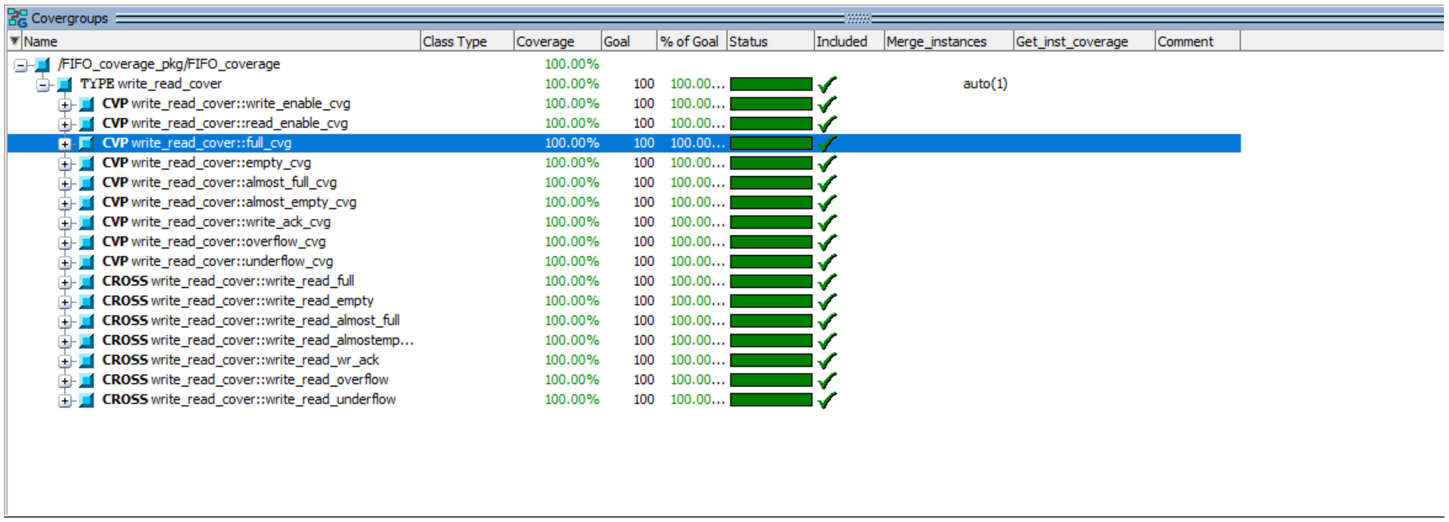
Toggles - by instance (/FIFO_top/MONITOR/FIFO_if)

```
sim:/FIFO_top/MONITOR/FIFO_if
  ✓ almostempty
  ✓ almostfull
  ✓ clk
  ✓ data_in
  ✓ data_out
  ✓ empty
  ✓ full
  ✓ overflow
  ✓ rd_en
  ✓ rst_n
  ✓ underflow
  ✓ wr_ack
  ✓ wr_en
```



```
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         86        86         0   100.00%

================================Toggle Details================================

Toggle Coverage for instance /FIFO_top/FIFO_if --

                                              Node      1H->0L      0L->1H  "Coverage"
                                              ---------------------------------------
                                       almostempty           1           1      100.00
                                        almostfull           1           1      100.00
                                               clk           1           1      100.00
                                     data_in[15-0]           1           1      100.00
                                    data_out[15-0]           1           1      100.00
                                             empty           1           1      100.00
                                              full           1           1      100.00
                                          overflow           1           1      100.00
                                             rd_en           1           1      100.00
                                             rst_n           1           1      100.00
                                         underflow           1           1      100.00
                                            wr_ack           1           1      100.00
                                             wr_en           1           1      100.00

Total Node Count     =         43
Toggled Node Count   =         43
Untoggled Node Count =          0

Toggle Coverage      =     100.00% (86 of 86 bins)
```

⇨ ***Functional coverage:***

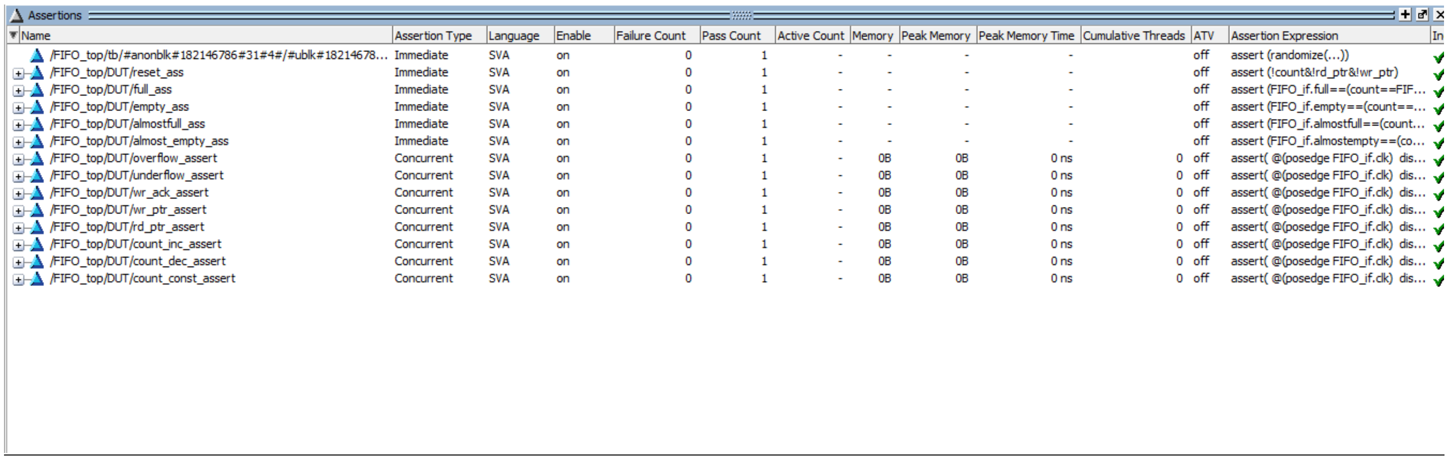


Covergroups

Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment
/FIFO_coverage_pkg/FIFO_coverage		100.00%							
TYPE write_read_cover		100.00%	100	100.00...	🟩	✓		auto(1)	
CVP write_read_cover::write_enable_cvg		100.00%	100	100.00...	🟩	✓			
CVP write_read_cover::read_enable_cvg		100.00%	100	100.00...	🟩	✓			
CVP write_read_cover::full_cvg		100.00%	100	100.00...	🟩	✓			
CVP write_read_cover::empty_cvg		100.00%	100	100.00...	🟩	✓			
CVP write_read_cover::almost_full_cvg		100.00%	100	100.00...	🟩	✓			
CVP write_read_cover::almost_empty_cvg		100.00%	100	100.00...	🟩	✓			
CVP write_read_cover::write_ack_cvg		100.00%	100	100.00...	🟩	✓			
CVP write_read_cover::overflow_cvg		100.00%	100	100.00...	🟩	✓			
CVP write_read_cover::underflow_cvg		100.00%	100	100.00...	🟩	✓			
CROSS write_read_cover::write_read_full		100.00%	100	100.00...	🟩	✓			
CROSS write_read_cover::write_read_empty		100.00%	100	100.00...	🟩	✓			
CROSS write_read_cover::write_read_almost_full		100.00%	100	100.00...	🟩	✓			
CROSS write_read_cover::write_read_almostemp...		100.00%	100	100.00...	🟩	✓			
CROSS write_read_cover::write_read_wr_ack		100.00%	100	100.00...	🟩	✓			
CROSS write_read_cover::write_read_overflow		100.00%	100	100.00...	🟩	✓			
CROSS write_read_cover::write_read_underflow		100.00%	100	100.00...	🟩	✓			

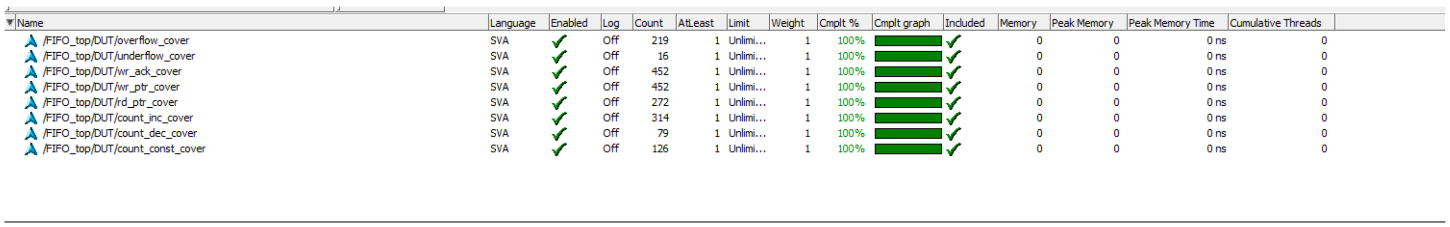
```
Directive Coverage:
    Directives                    8         8         0    100.00%

DIRECTIVE COVERAGE:
--------------------------------------------------------------------------------------------
Name                                     Design Design   Lang File(Line)      Hits Status
                                         Unit   UnitType
--------------------------------------------------------------------------------------------
/FIFO_top/DUT/overflow_cover             FIFO   Verilog  SVA  FIFO.sv(146)     219 Covered
/FIFO_top/DUT/underflow_cover            FIFO   Verilog  SVA  FIFO.sv(147)      16 Covered
/FIFO_top/DUT/wr_ack_cover               FIFO   Verilog  SVA  FIFO.sv(148)     452 Covered
/FIFO_top/DUT/wr_ptr_cover               FIFO   Verilog  SVA  FIFO.sv(149)     452 Covered
/FIFO_top/DUT/rd_ptr_cover               FIFO   Verilog  SVA  FIFO.sv(150)     272 Covered
/FIFO_top/DUT/count_inc_cover            FIFO   Verilog  SVA  FIFO.sv(151)     314 Covered
/FIFO_top/DUT/count_dec_cover            FIFO   Verilog  SVA  FIFO.sv(152)      79 Covered
/FIFO_top/DUT/count_const_cover          FIFO   Verilog  SVA  FIFO.sv(153)     126 Covered

=================================================================================
=== Instance: /FIFO_coverage_pkg
=== Design Unit: work.FIFO_coverage_pkg
=================================================================================

Covergroup Coverage:
    Covergroups                   1        na        na    100.00%
        Coverpoints/Crosses      16        na        na        na
            Covergroup Bins      66        66         0    100.00%
```

⇨ ***Assertions:***



Assertions

Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Count	Memory	Peak Memory	Peak Memory Time	Cumulative Threads	ATV	Assertion Expression	In
/FIFO_top/tb/#anonblk#182146786#31#4#/#ublk#18214678...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/FIFO_top/DUT/reset_ass	Immediate	SVA	on	0	1	-	-	-	-		off	assert (!count&!rd_ptr&!wr_ptr)	✓
/FIFO_top/DUT/full_ass	Immediate	SVA	on	0	1	-	-	-	-		off	assert (FIFO_if.full==(count==FIF...	✓
/FIFO_top/DUT/empty_ass	Immediate	SVA	on	0	1	-	-	-	-		off	assert (FIFO_if.empty==(count==...	✓
/FIFO_top/DUT/almostfull_ass	Immediate	SVA	on	0	1	-	-	-	-		off	assert (FIFO_if.almostfull==(count...	✓
/FIFO_top/DUT/almost_empty_ass	Immediate	SVA	on	0	1	-	-	-	-		off	assert (FIFO_if.almostempty==(co...	✓
/FIFO_top/DUT/overflow_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.clk) dis...	✓
/FIFO_top/DUT/underflow_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.clk) dis...	✓
/FIFO_top/DUT/wr_ack_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.clk) dis...	✓
/FIFO_top/DUT/wr_ptr_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.clk) dis...	✓
/FIFO_top/DUT/rd_ptr_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.clk) dis...	✓
/FIFO_top/DUT/count_inc_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.clk) dis...	✓
/FIFO_top/DUT/count_dec_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.clk) dis...	✓
/FIFO_top/DUT/count_const_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.clk) dis...	✓



Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
/FIFO_top/DUT/overflow_cover	SVA	✓	Off	219	1	Unlimi...	1	100%	🟩	✓	0	0	0 ns	0
/FIFO_top/DUT/underflow_cover	SVA	✓	Off	16	1	Unlimi...	1	100%	🟩	✓	0	0	0 ns	0
/FIFO_top/DUT/wr_ack_cover	SVA	✓	Off	452	1	Unlimi...	1	100%	🟩	✓	0	0	0 ns	0
/FIFO_top/DUT/wr_ptr_cover	SVA	✓	Off	452	1	Unlimi...	1	100%	🟩	✓	0	0	0 ns	0
/FIFO_top/DUT/rd_ptr_cover	SVA	✓	Off	272	1	Unlimi...	1	100%	🟩	✓	0	0	0 ns	0
/FIFO_top/DUT/count_inc_cover	SVA	✓	Off	314	1	Unlimi...	1	100%	🟩	✓	0	0	0 ns	0
/FIFO_top/DUT/count_dec_cover	SVA	✓	Off	79	1	Unlimi...	1	100%	🟩	✓	0	0	0 ns	0
/FIFO_top/DUT/count_const_cover	SVA	✓	Off	126	1	Unlimi...	1	100%	🟩	✓	0	0	0 ns	0