Final Project

Project (1)-FIFO:

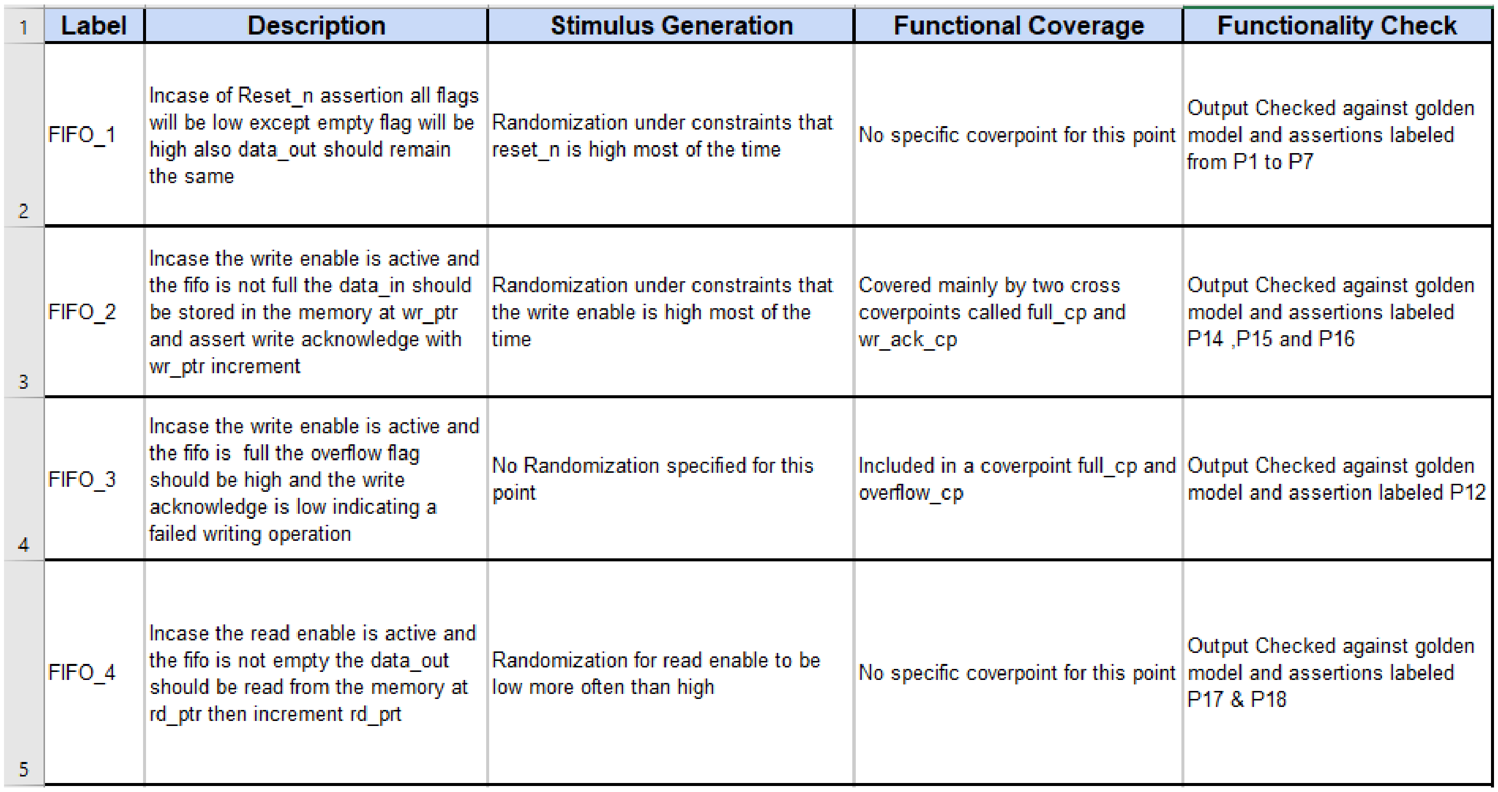
1. Verification plan

a) Asynchronous Reset functionality.
b) Testing writing operation by checking memory.
c) Testing reading operation by monitoring the data_out.
d) Testing if no operation if the data_out stayed stable or not
e) Testing the combinational flags assertions and de-assertions in the specified scenarios for each one.

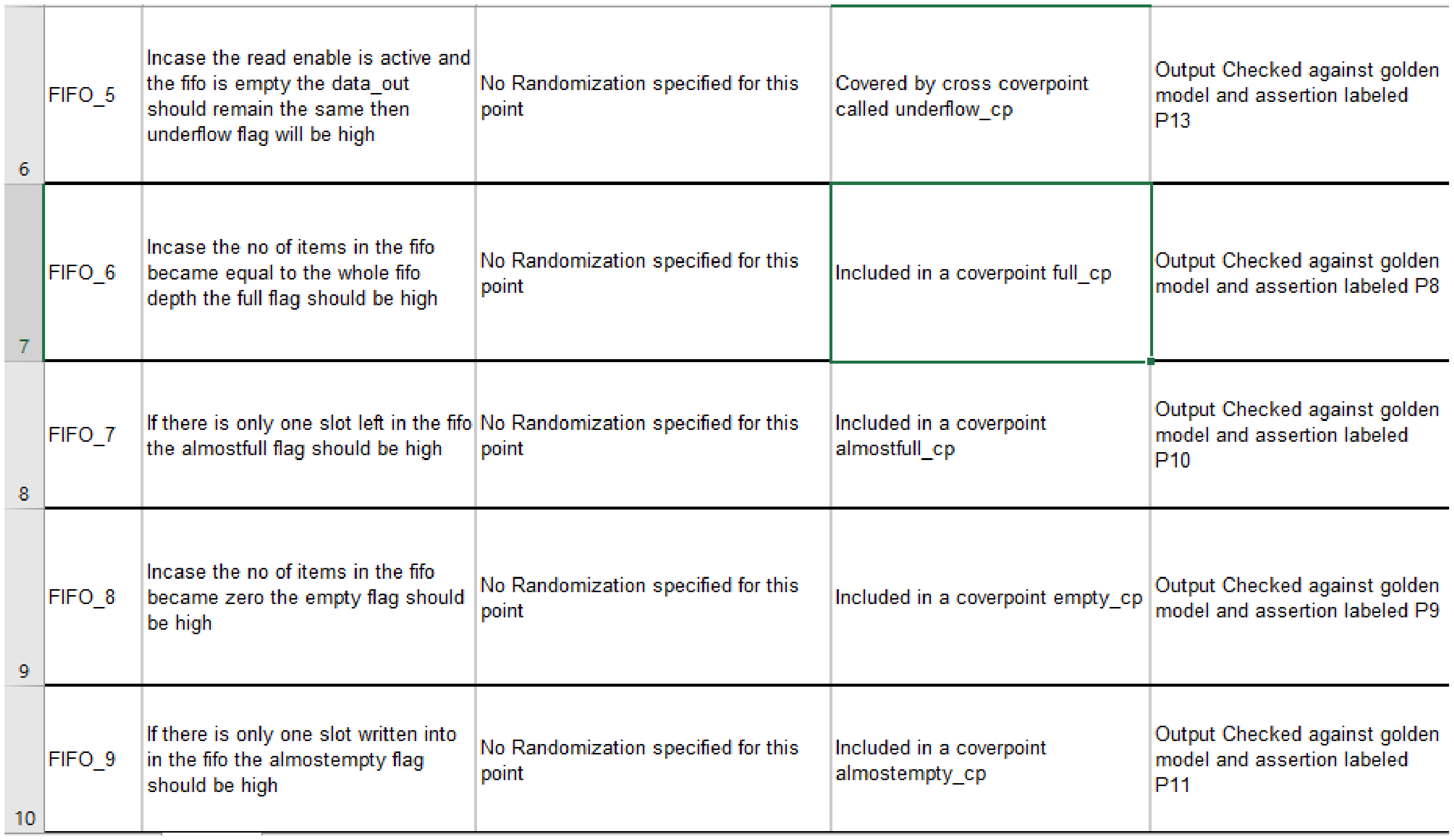
Note:

Inside coverage class triggering coverage process takes place only at high reset to avoid meaningless coverage.

2. Verification Document

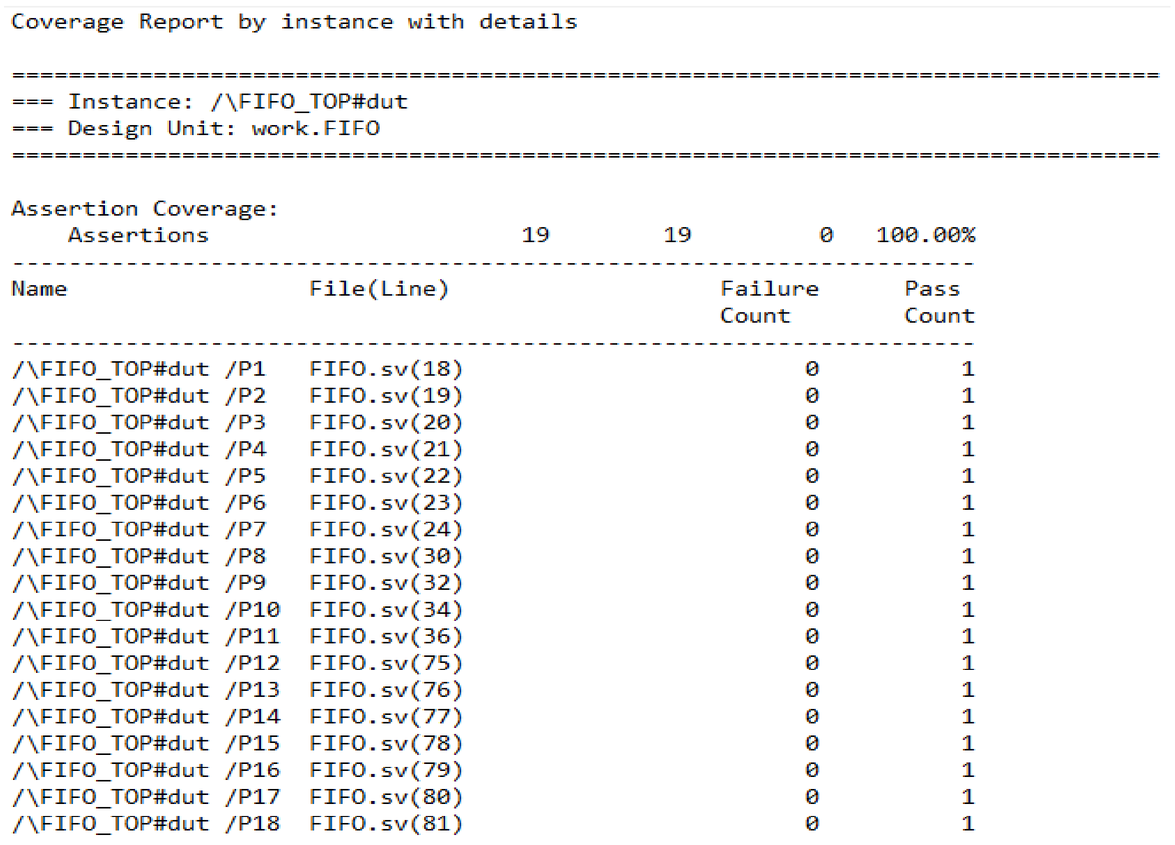


Label	Description	Stimulus Generation	Functional Coverage	Functionality Check
FIFO_1	Incase of Reset_n assertion all flags will be low except empty flag will be high also data_out should remain the same	Randomization under constraints that reset_n is high most of the time	No specific coverpoint for this point	Output Checked against golden model and assertions labeled from P1 to P7
FIFO_2	Incase the write enable is active and the fifo is not full the data_in should be stored in the memory at wr_ptr and assert write acknowledge with wr_ptr increment	Randomization under constraints that the write enable is high most of the time	Covered mainly by two cross coverpoints called full_cp and wr_ack_cp	Output Checked against golden model and assertions labeled P14 ,P15 and P16
FIFO_3	Incase the write enable is active and the fifo is full the overflow flag should be high and the write acknowledge is low indicating a failed writing operation	No Randomization specified for this point	Included in a coverpoint full_cp and overflow_cp	Output Checked against golden model and assertion labeled P12
FIFO_4	Incase the read enable is active and the fifo is not empty the data_out should be read from the memory at rd_ptr then increment rd_prt	Randomization for read enable to be low more often than high	No specific coverpoint for this point	Output Checked against golden model and assertions labeled P17 & P18



6	FIFO_5	Incase the read enable is active and the fifo is empty the data_out should remain the same then underflow flag will be high	No Randomization specified for this point	Covered by cross coverpoint called underflow_cp	Output Checked against golden model and assertion labeled P13
7	FIFO_6	Incase the no of items in the fifo became equal to the whole fifo depth the full flag should be high	No Randomization specified for this point	Included in a coverpoint full_cp	Output Checked against golden model and assertion labeled P8
8	FIFO_7	If there is only one slot left in the fifo the almostfull flag should be high	No Randomization specified for this point	Included in a coverpoint almostfull_cp	Output Checked against golden model and assertion labeled P10
9	FIFO_8	Incase the no of items in the fifo became zero the empty flag should be high	No Randomization specified for this point	Included in a coverpoint empty_cp	Output Checked against golden model and assertion labeled P9
10	FIFO_9	If there is only one slot written into in the fifo the almostempty flag should be high	No Randomization specified for this point	Included in a coverpoint almostempty_cp	Output Checked against golden model and assertion labeled P11

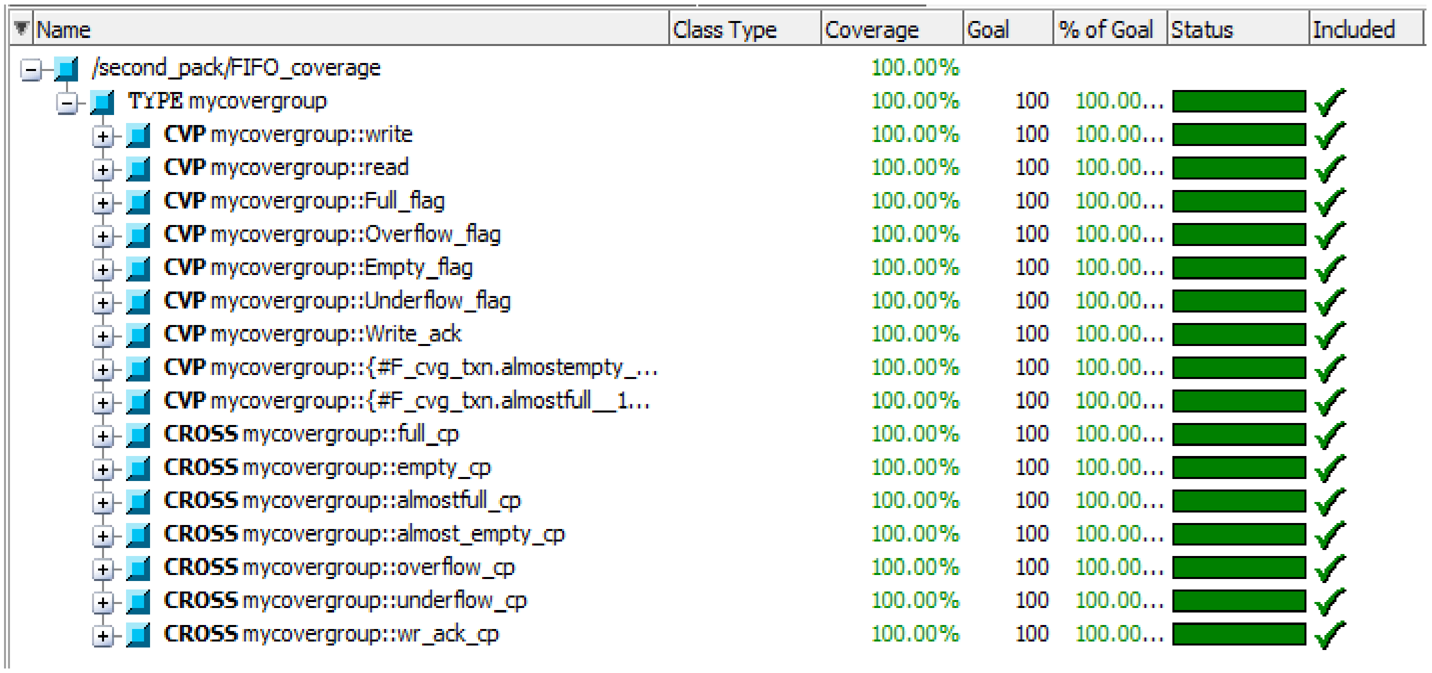
3. Coverage reports:



```
Coverage Report by instance with details

=================================================================================
=== Instance: /\FIFO_TOP#dut
=== Design Unit: work.FIFO
=================================================================================

Assertion Coverage:
    Assertions                         19         19          0    100.00%
--------------------------------------------------------------------
Name                    File(Line)                    Failure      Pass
                                                      Count        Count
--------------------------------------------------------------------
/\FIFO_TOP#dut /P1     FIFO.sv(18)                         0           1
/\FIFO_TOP#dut /P2     FIFO.sv(19)                         0           1
/\FIFO_TOP#dut /P3     FIFO.sv(20)                         0           1
/\FIFO_TOP#dut /P4     FIFO.sv(21)                         0           1
/\FIFO_TOP#dut /P5     FIFO.sv(22)                         0           1
/\FIFO_TOP#dut /P6     FIFO.sv(23)                         0           1
/\FIFO_TOP#dut /P7     FIFO.sv(24)                         0           1
/\FIFO_TOP#dut /P8     FIFO.sv(30)                         0           1
/\FIFO_TOP#dut /P9     FIFO.sv(32)                         0           1
/\FIFO_TOP#dut /P10    FIFO.sv(34)                         0           1
/\FIFO_TOP#dut /P11    FIFO.sv(36)                         0           1
/\FIFO_TOP#dut /P12    FIFO.sv(75)                         0           1
/\FIFO_TOP#dut /P13    FIFO.sv(76)                         0           1
/\FIFO_TOP#dut /P14    FIFO.sv(77)                         0           1
/\FIFO_TOP#dut /P15    FIFO.sv(78)                         0           1
/\FIFO_TOP#dut /P16    FIFO.sv(79)                         0           1
/\FIFO_TOP#dut /P17    FIFO.sv(80)                         0           1
/\FIFO_TOP#dut /P18    FIFO.sv(81)                         0           1
```

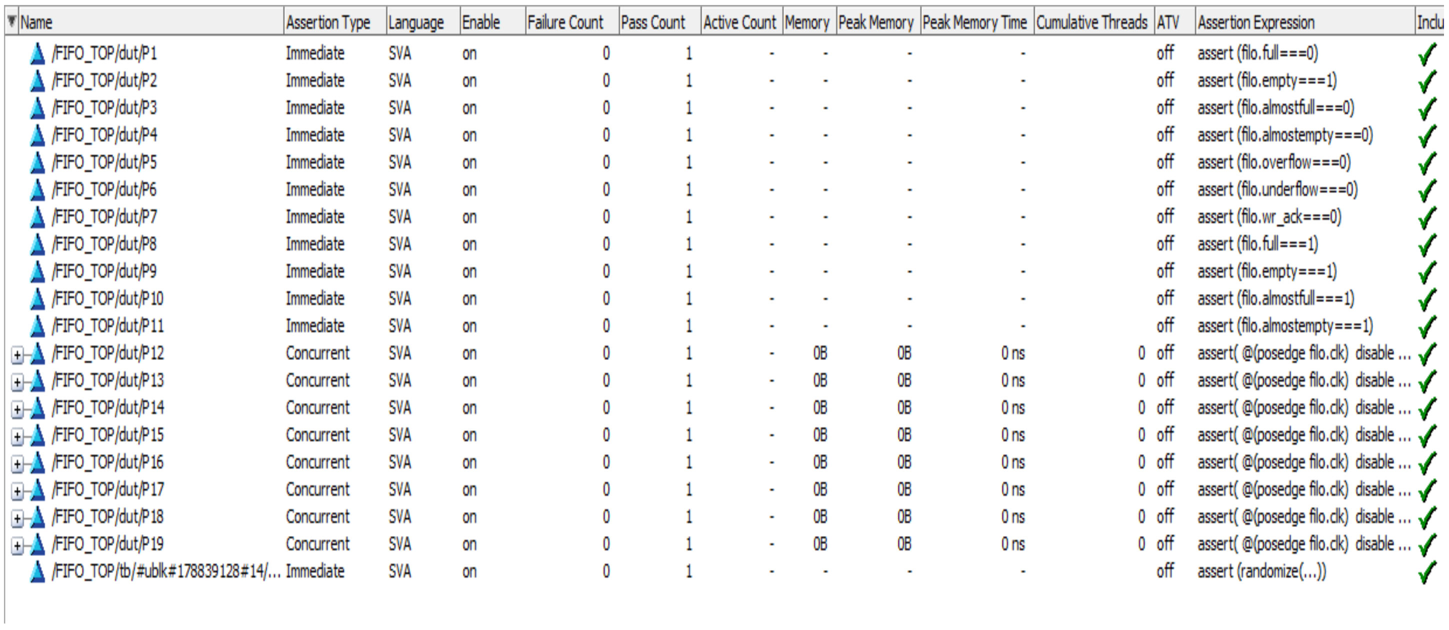
```
    Branch Coverage:
        Enabled Coverage              Bins      Hits    Misses  Coverage
        ----------------              ----      ----    ------  --------
        Branches                        41        41         0   100.00%

    ================================Branch Details================================


    Condition Coverage:
        Enabled Coverage              Bins   Covered    Misses  Coverage
        ----------------              ----      ----    ------  --------
        Conditions                      16        16         0   100.00%

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



Directive Coverage:
    Directives                      8         8         0   100.00%

DIRECTIVE COVERAGE:
-------------------------------------------------------------------------------------------
Name                                    Design Design   Lang File(Line)      Hits Status
                                        Unit   UnitType
-------------------------------------------------------------------------------------------
/\FIFO_TOP#dut /C12                     FIFO   Verilog  SVA  FIFO.sv(84)     1247 Covered
/\FIFO_TOP#dut /C13                     FIFO   Verilog  SVA  FIFO.sv(85)     2031 Covered
/\FIFO_TOP#dut /C14                     FIFO   Verilog  SVA  FIFO.sv(86)     3483 Covered
/\FIFO_TOP#dut /C15                     FIFO   Verilog  SVA  FIFO.sv(87)     1260 Covered
/\FIFO_TOP#dut /C16                     FIFO   Verilog  SVA  FIFO.sv(88)     1260 Covered
/\FIFO_TOP#dut /C17                     FIFO   Verilog  SVA  FIFO.sv(89)      427 Covered
/\FIFO_TOP#dut /C18                     FIFO   Verilog  SVA  FIFO.sv(90)      427 Covered
/\FIFO_TOP#dut /C19                     FIFO   Verilog  SVA  FIFO.sv(91)     1769 Covered
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                      30        30         0   100.00%

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

   Toggle Coverage:
       Enabled Coverage              Bins      Hits    Misses  Coverage
       ----------------              ----      ----    ------  --------
       Toggles                         12        12         0   100.00%

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

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

➢ Function Coverage:



Name	Class Type	Coverage	Goal	% of Goal	Status	Included
/second_pack/FIFO_coverage		100.00%				
TYPE mycovergroup		100.00%	100	100.00...	🟩	✓
CVP mycovergroup::write		100.00%	100	100.00...	🟩	✓
CVP mycovergroup::read		100.00%	100	100.00...	🟩	✓
CVP mycovergroup::Full_flag		100.00%	100	100.00...	🟩	✓
CVP mycovergroup::Overflow_flag		100.00%	100	100.00...	🟩	✓
CVP mycovergroup::Empty_flag		100.00%	100	100.00...	🟩	✓
CVP mycovergroup::Underflow_flag		100.00%	100	100.00...	🟩	✓
CVP mycovergroup::Write_ack		100.00%	100	100.00...	🟩	✓
CVP mycovergroup::{#F_cvg_txn.almostempty_...		100.00%	100	100.00...	🟩	✓
CVP mycovergroup::{#F_cvg_txn.almostfull__1...		100.00%	100	100.00...	🟩	✓
CROSS mycovergroup::full_cp		100.00%	100	100.00...	🟩	✓
CROSS mycovergroup::empty_cp		100.00%	100	100.00...	🟩	✓
CROSS mycovergroup::almostfull_cp		100.00%	100	100.00...	🟩	✓
CROSS mycovergroup::almost_empty_cp		100.00%	100	100.00...	🟩	✓
CROSS mycovergroup::overflow_cp		100.00%	100	100.00...	🟩	✓
CROSS mycovergroup::underflow_cp		100.00%	100	100.00...	🟩	✓
CROSS mycovergroup::wr_ack_cp		100.00%	100	100.00...	🟩	✓

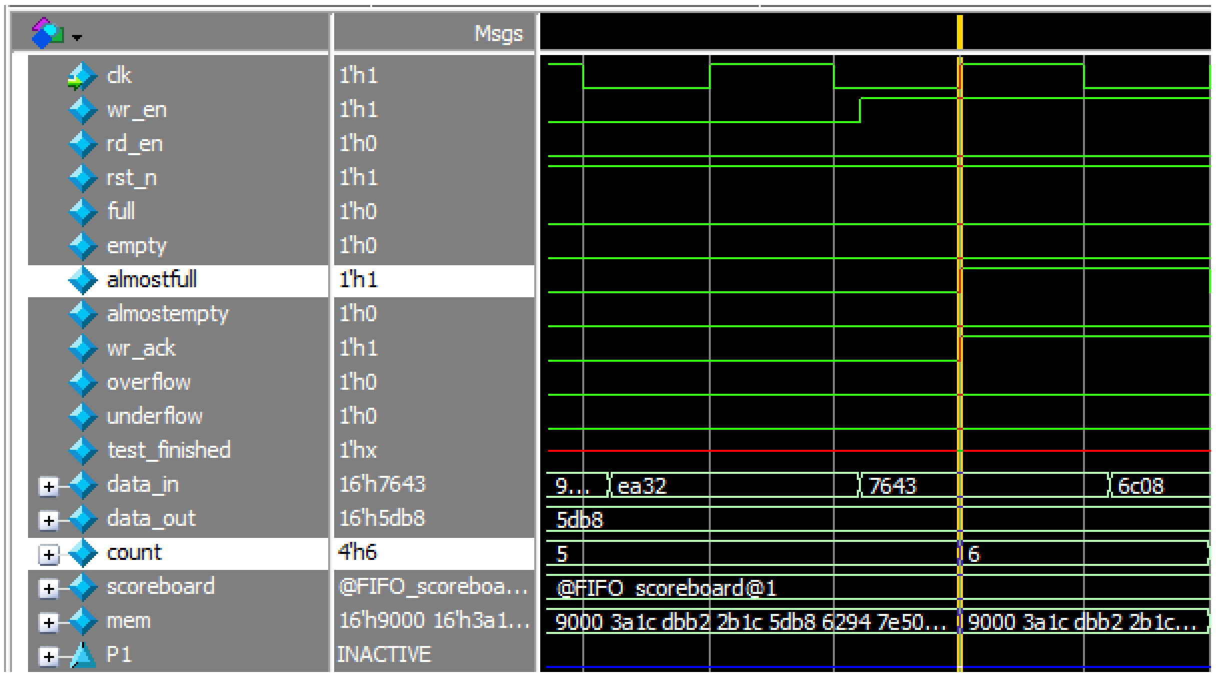
➢ Assertion Coverage:



Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Count	Memory	Peak Memory	Peak Memory Time	Cumulative Threads	ATV	Assertion Expression	Inclu
/FIFO_TOP/dut/P1	Immediate	SVA	on	0	1	-	-	-	-		off	assert (filo.full===0)	✓
/FIFO_TOP/dut/P2	Immediate	SVA	on	0	1	-	-	-	-		off	assert (filo.empty===1)	✓
/FIFO_TOP/dut/P3	Immediate	SVA	on	0	1	-	-	-	-		off	assert (filo.almostfull===0)	✓
/FIFO_TOP/dut/P4	Immediate	SVA	on	0	1	-	-	-	-		off	assert (filo.almostempty===0)	✓
/FIFO_TOP/dut/P5	Immediate	SVA	on	0	1	-	-	-	-		off	assert (filo.overflow===0)	✓
/FIFO_TOP/dut/P6	Immediate	SVA	on	0	1	-	-	-	-		off	assert (filo.underflow===0)	✓
/FIFO_TOP/dut/P7	Immediate	SVA	on	0	1	-	-	-	-		off	assert (filo.wr_ack===0)	✓
/FIFO_TOP/dut/P8	Immediate	SVA	on	0	1	-	-	-	-		off	assert (filo.full===1)	✓
/FIFO_TOP/dut/P9	Immediate	SVA	on	0	1	-	-	-	-		off	assert (filo.empty===1)	✓
/FIFO_TOP/dut/P10	Immediate	SVA	on	0	1	-	-	-	-		off	assert (filo.almostfull===1)	✓
/FIFO_TOP/dut/P11	Immediate	SVA	on	0	1	-	-	-	-		off	assert (filo.almostempty===1)	✓
/FIFO_TOP/dut/P12	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge filo.clk) disable ...	✓
/FIFO_TOP/dut/P13	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge filo.clk) disable ...	✓
/FIFO_TOP/dut/P14	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge filo.clk) disable ...	✓
/FIFO_TOP/dut/P15	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge filo.clk) disable ...	✓
/FIFO_TOP/dut/P16	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge filo.clk) disable ...	✓
/FIFO_TOP/dut/P17	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge filo.clk) disable ...	✓
/FIFO_TOP/dut/P18	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge filo.clk) disable ...	✓
/FIFO_TOP/dut/P19	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge filo.clk) disable ...	✓
/FIFO_TOP/tb/#ublk#178839128#14/...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓

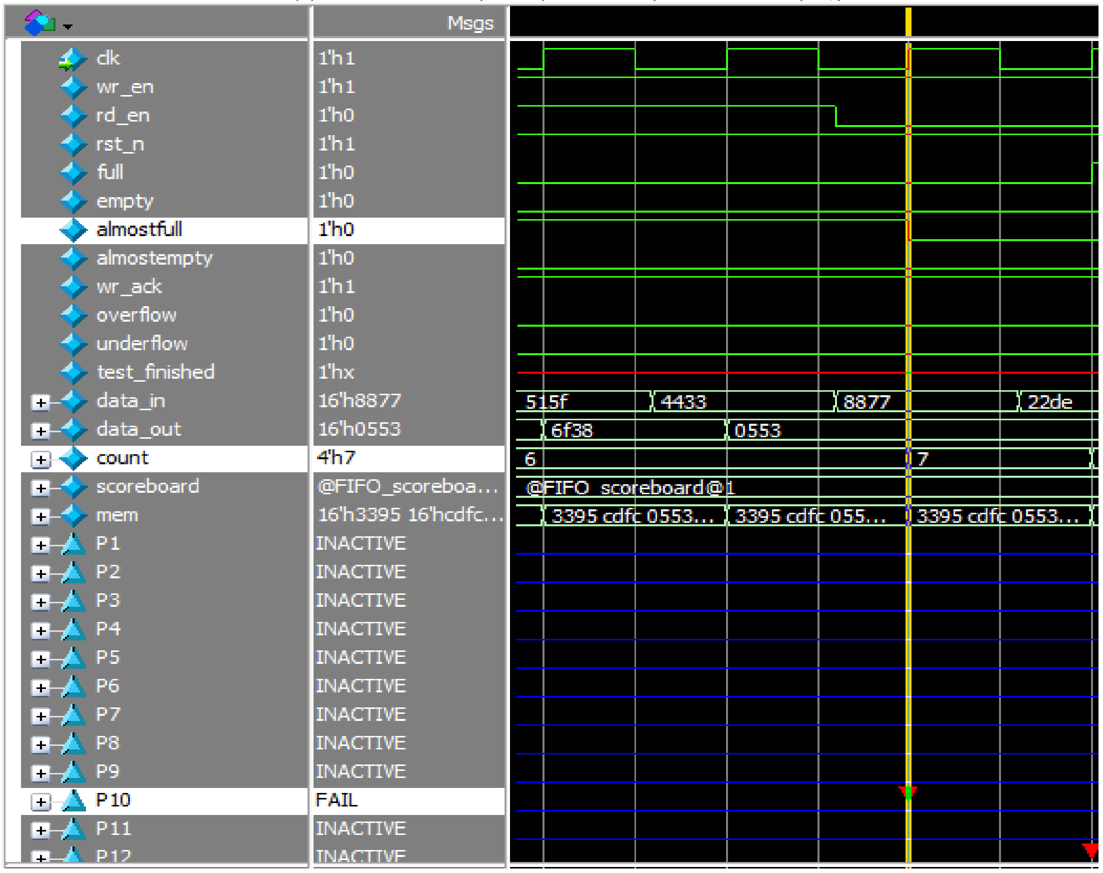
4. Bugs report:

a) Almostfull flag is assigned high when count = (depth-2) not depth-1 causing it to be high way too early.



	Msgs
clk	1'h1
wr_en	1'h1
rd_en	1'h0
rst_n	1'h1
full	1'h0
empty	1'h0
almostfull	1'h1
almostempty	1'h0
wr_ack	1'h1
overflow	1'h0
underflow	1'h0
test_finished	1'hx
data_in	16'h7643
data_out	16'h5db8
count	4'h6
scoreboard	@FIFO_scoreboa...
mem	16'h9000 16'h3a1...
P1	INACTIVE

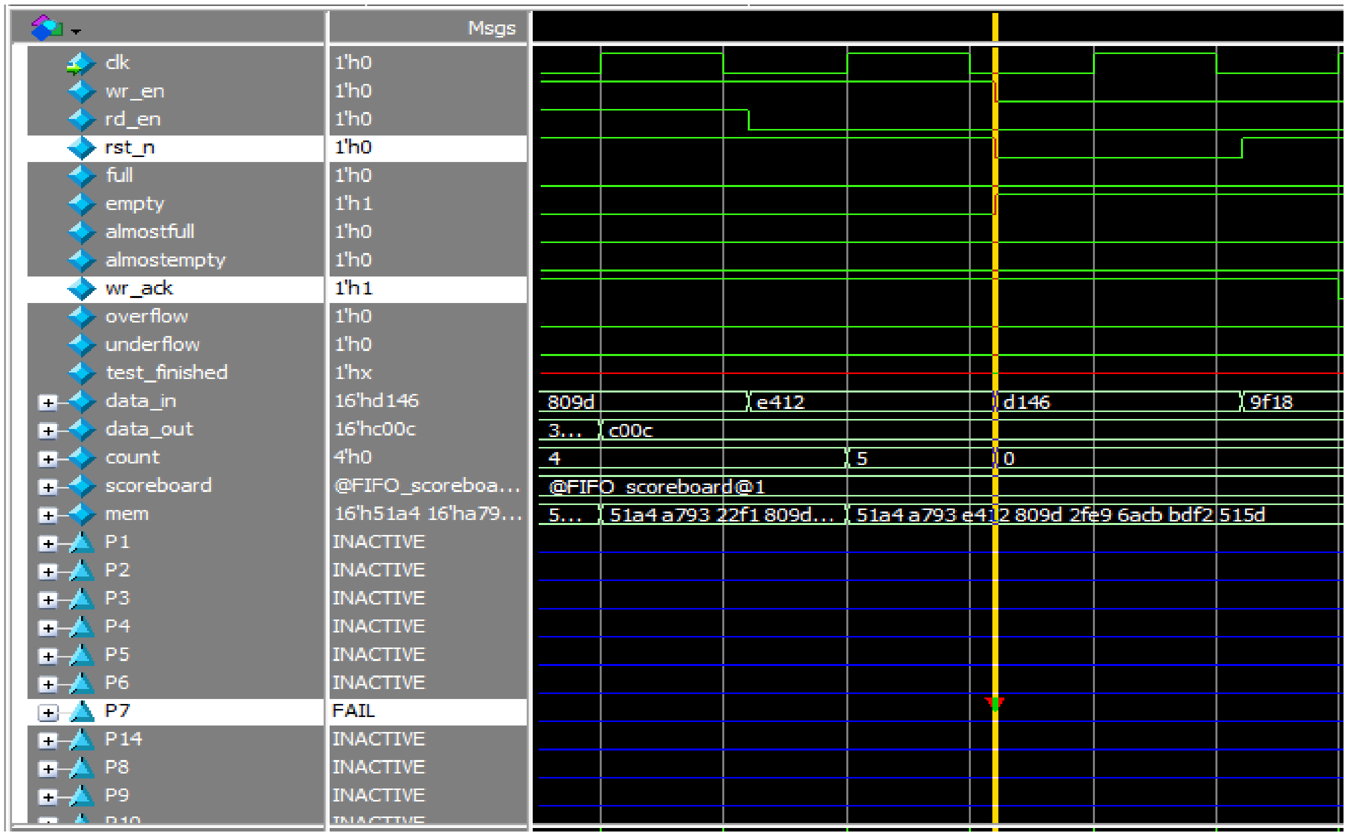
Snippet shows the false positive of the almost flag



	Msgs
clk	1'h1
wr_en	1'h1
rd_en	1'h0
rst_n	1'h1
full	1'h0
empty	1'h0
almostfull	1'h0
almostempty	1'h0
wr_ack	1'h1
overflow	1'h0
underflow	1'h0
test_finished	1'hx
data_in	16'h8877
data_out	16'h0553
count	4'h7
scoreboard	@FIFO_scoreboa...
mem	16'h3395 16'hcdfc...
P1	INACTIVE
P2	INACTIVE
P3	INACTIVE
P4	INACTIVE
P5	INACTIVE
P6	INACTIVE
P7	INACTIVE
P8	INACTIVE
P9	INACTIVE
P10	FAIL
P11	INACTIVE
P12	INACTIVE

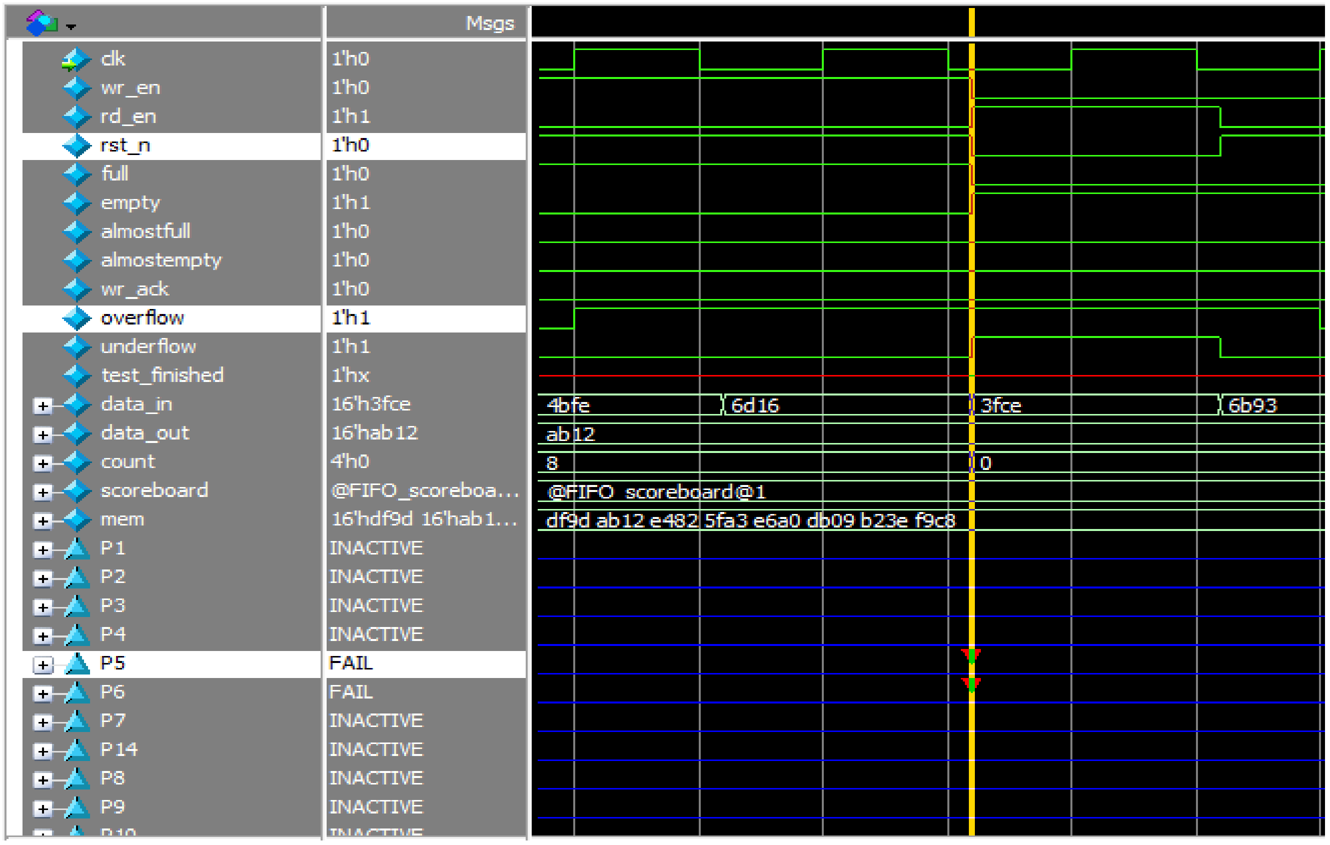
Snippet shows the false negative of the almost flag

b) During asserted reset if the previous write acknowledge flag was high, it remains high giving a false high signal.



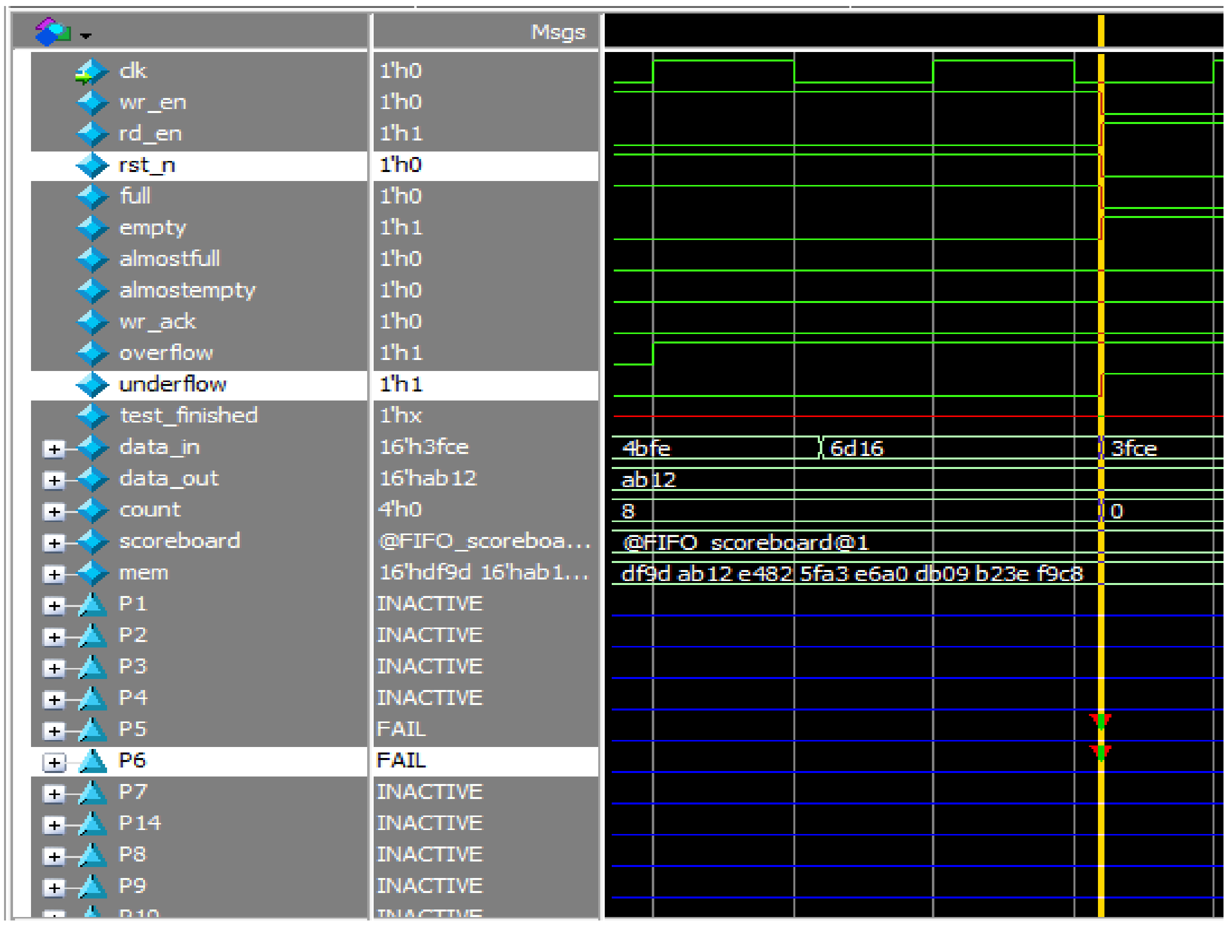
	Msgs
clk	1'h0
wr_en	1'h0
rd_en	1'h0
rst_n	1'h0
full	1'h0
empty	1'h1
almostfull	1'h0
almostempty	1'h0
wr_ack	1'h1
overflow	1'h0
underflow	1'h0
test_finished	1'hx
data_in	16'hd146
data_out	16'hc00c
count	4'h0
scoreboard	@FIFO_scoreboa...
mem	16'h51a4 16'ha79...
P1	INACTIVE
P2	INACTIVE
P3	INACTIVE
P4	INACTIVE
P5	INACTIVE
P6	INACTIVE
P7	FAIL
P14	INACTIVE
P8	INACTIVE
P9	INACTIVE

c) Also, before asserting reset if overflow flag was high, it will remain unchanged after asserting reset.



	Msgs
clk	1'h0
wr_en	1'h0
rd_en	1'h1
rst_n	1'h0
full	1'h0
empty	1'h1
almostfull	1'h0
almostempty	1'h0
wr_ack	1'h0
overflow	1'h1
underflow	1'h1
test_finished	1'hx
data_in	16'h3fce
data_out	16'hab12
count	4'h0
scoreboard	@FIFO_scoreboa...
mem	16'hdf9d 16'hab1...
P1	INACTIVE
P2	INACTIVE
P3	INACTIVE
P4	INACTIVE
P5	FAIL
P6	FAIL
P7	INACTIVE
P14	INACTIVE
P8	INACTIVE
P9	INACTIVE

d) During active reset if the read enable signal is high then the underflow flag will be falsely high.



	Msgs
clk	1'h0
wr_en	1'h0
rd_en	1'h1
rst_n	1'h0
full	1'h0
empty	1'h1
almostfull	1'h0
almostempty	1'h0
wr_ack	1'h0
overflow	1'h1
underflow	1'h1
test_finished	1'hx
data_in	16'h3fce
data_out	16'hab12
count	4'h0
scoreboard	@FIFO_scoreboa...
mem	16'hdf9d 16'hab1...
P1	INACTIVE
P2	INACTIVE
P3	INACTIVE
P4	INACTIVE
P5	FAIL
P6	FAIL
P7	INACTIVE
P14	INACTIVE
P8	INACTIVE
P9	INACTIVE

e) Incase FIFO is full and both of the write and read enable are high from my understanding of the specs it should reject write operation because it is full (wr_ack =0, overflow=1) but it will perform the read operation as the two operations run in parallel.

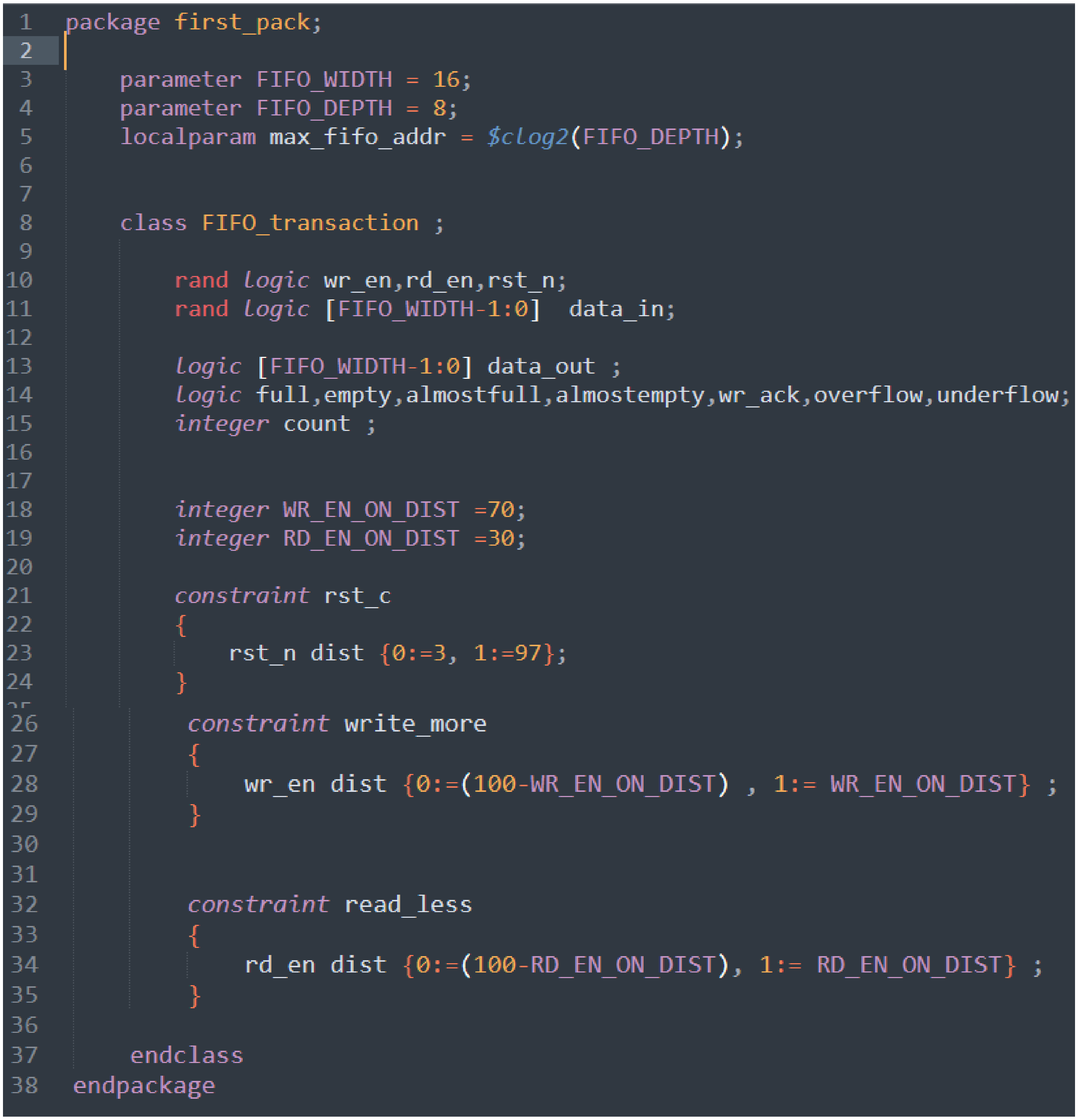
f) Incase FIFO is empty and both of the write and read enable are high it should reject read operation because it is empty (underflow=1) but it will perform the write operation.

Note:

Combinational flags (full, almsotfull, almostempty, empty) implemented with assign statements which gets triggered when count is updated but the problem is that this count gets updated sequentially with the clock positive edge clock so as a result those flags are updated sequentially.

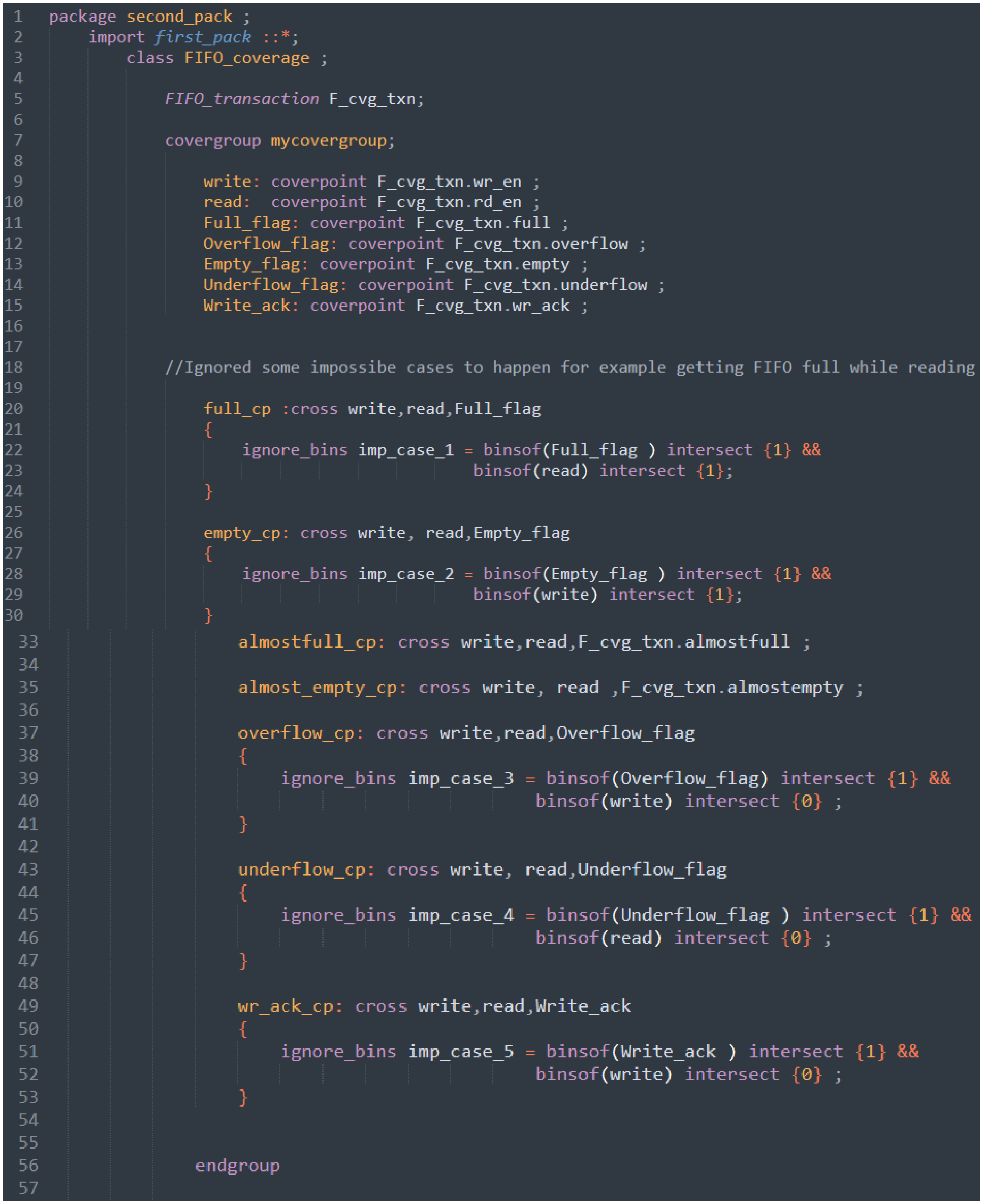
5. Code snippets

a) First Package:

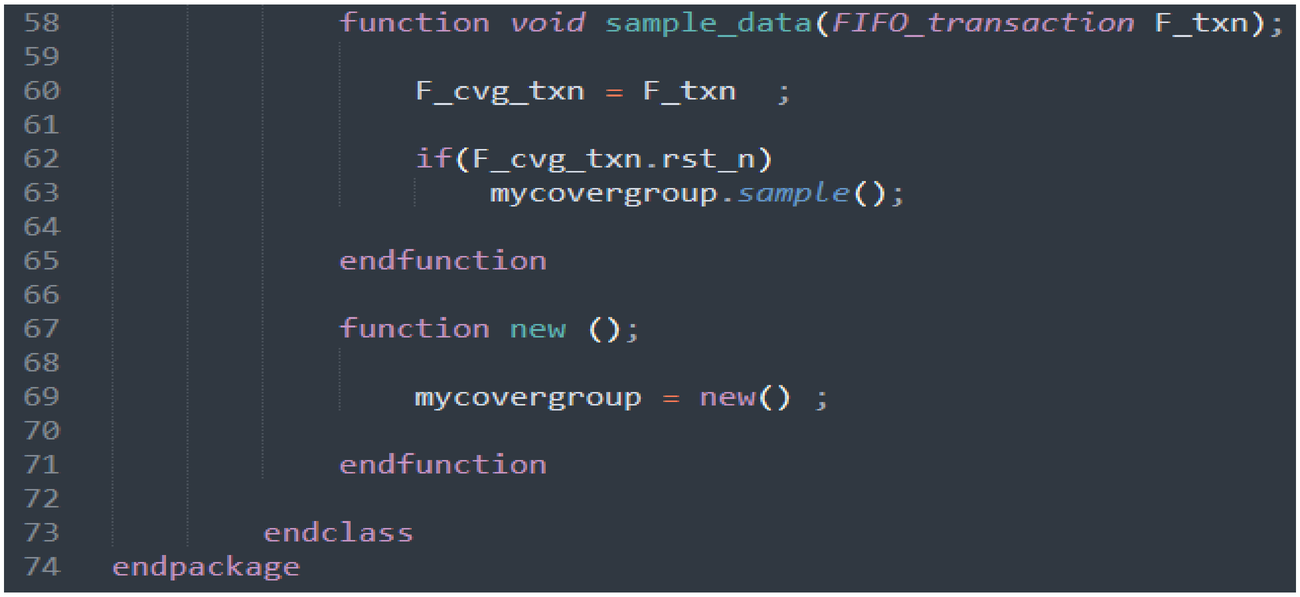


```
package first_pack;

    parameter FIFO_WIDTH = 16;
    parameter FIFO_DEPTH = 8;
    localparam max_fifo_addr = $clog2(FIFO_DEPTH);


    class FIFO_transaction ;

        rand logic wr_en,rd_en,rst_n;
        rand logic [FIFO_WIDTH-1:0]  data_in;

        logic [FIFO_WIDTH-1:0] data_out ;
        logic full,empty,almostfull,almostempty,wr_ack,overflow,underflow;
        integer count ;


        integer WR_EN_ON_DIST =70;
        integer RD_EN_ON_DIST =30;

        constraint rst_c
        {
            rst_n dist {0:=3, 1:=97};
        }

        constraint write_more
        {
            wr_en dist {0:=(100-WR_EN_ON_DIST) , 1:= WR_EN_ON_DIST} ;
        }


        constraint read_less
        {
            rd_en dist {0:=(100-RD_EN_ON_DIST), 1:= RD_EN_ON_DIST} ;
        }

    endclass
endpackage
```

b) Second package:

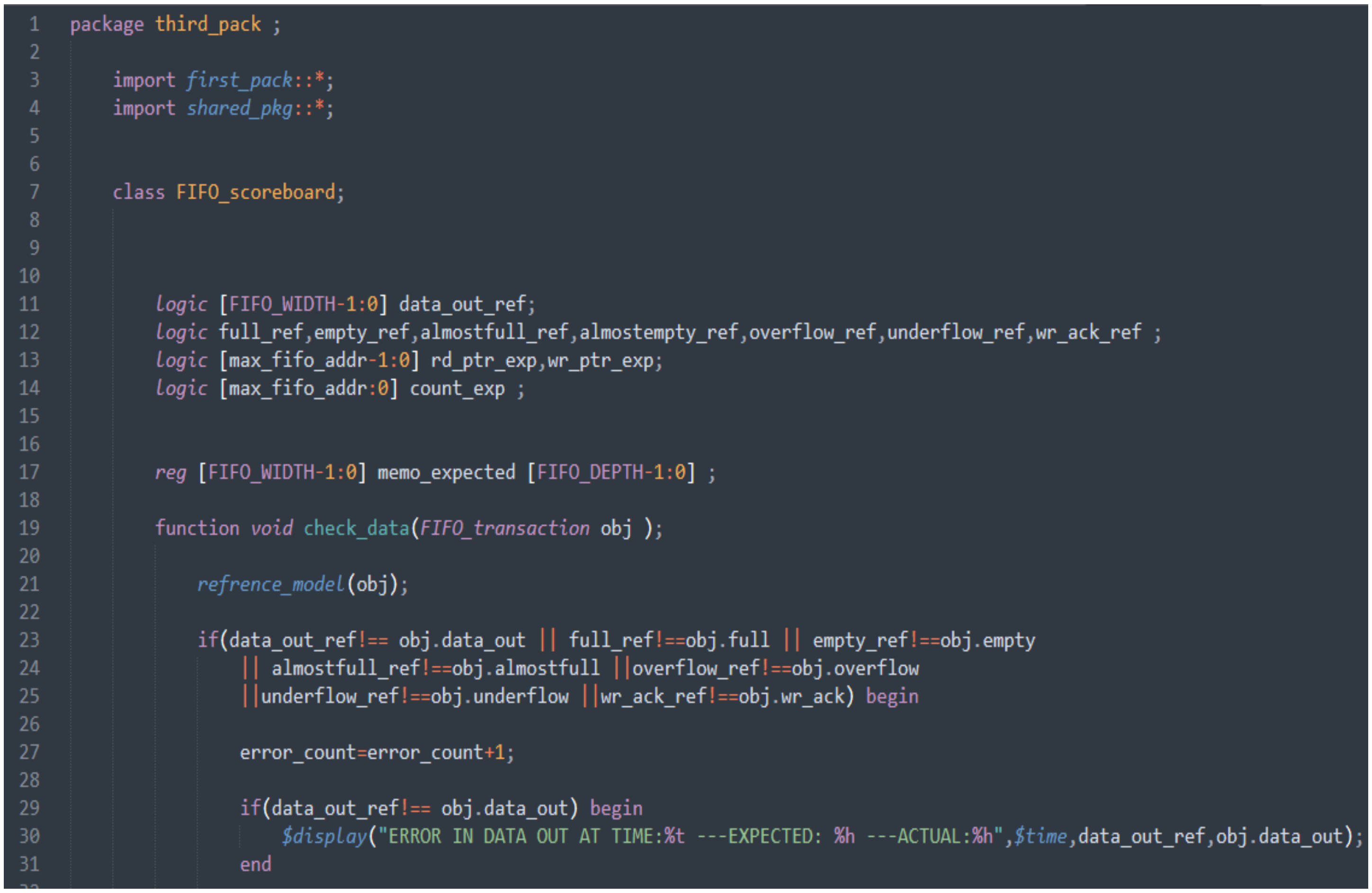


```
package second_pack ;
    import first_pack ::*;
        class FIFO_coverage ;

            FIFO_transaction F_cvg_txn;

            covergroup mycovergroup;

                write: coverpoint F_cvg_txn.wr_en ;
                read:  coverpoint F_cvg_txn.rd_en ;
                Full_flag: coverpoint F_cvg_txn.full ;
                Overflow_flag: coverpoint F_cvg_txn.overflow ;
                Empty_flag: coverpoint F_cvg_txn.empty ;
                Underflow_flag: coverpoint F_cvg_txn.underflow ;
                Write_ack: coverpoint F_cvg_txn.wr_ack ;


            //Ignored some impossibe cases to happen for example getting FIFO full while reading

                full_cp :cross write,read,Full_flag
                {
                    ignore_bins imp_case_1 = binsof(Full_flag ) intersect {1} &&
                                            binsof(read) intersect {1};
                }

                empty_cp: cross write, read,Empty_flag
                {
                    ignore_bins imp_case_2 = binsof(Empty_flag ) intersect {1} &&
                                            binsof(write) intersect {1};
                }
                almostfull_cp: cross write,read,F_cvg_txn.almostfull ;

                almost_empty_cp: cross write, read ,F_cvg_txn.almostempty ;

                overflow_cp: cross write,read,Overflow_flag
                {
                    ignore_bins imp_case_3 = binsof(Overflow_flag) intersect {1} &&
                                            binsof(write) intersect {0} ;
                }

                underflow_cp: cross write, read,Underflow_flag
                {
                    ignore_bins imp_case_4 = binsof(Underflow_flag ) intersect {1} &&
                                            binsof(read) intersect {0} ;
                }

                wr_ack_cp: cross write,read,Write_ack
                {
                    ignore_bins imp_case_5 = binsof(Write_ack ) intersect {1} &&
                                            binsof(write) intersect {0} ;
                }


            endgroup

```

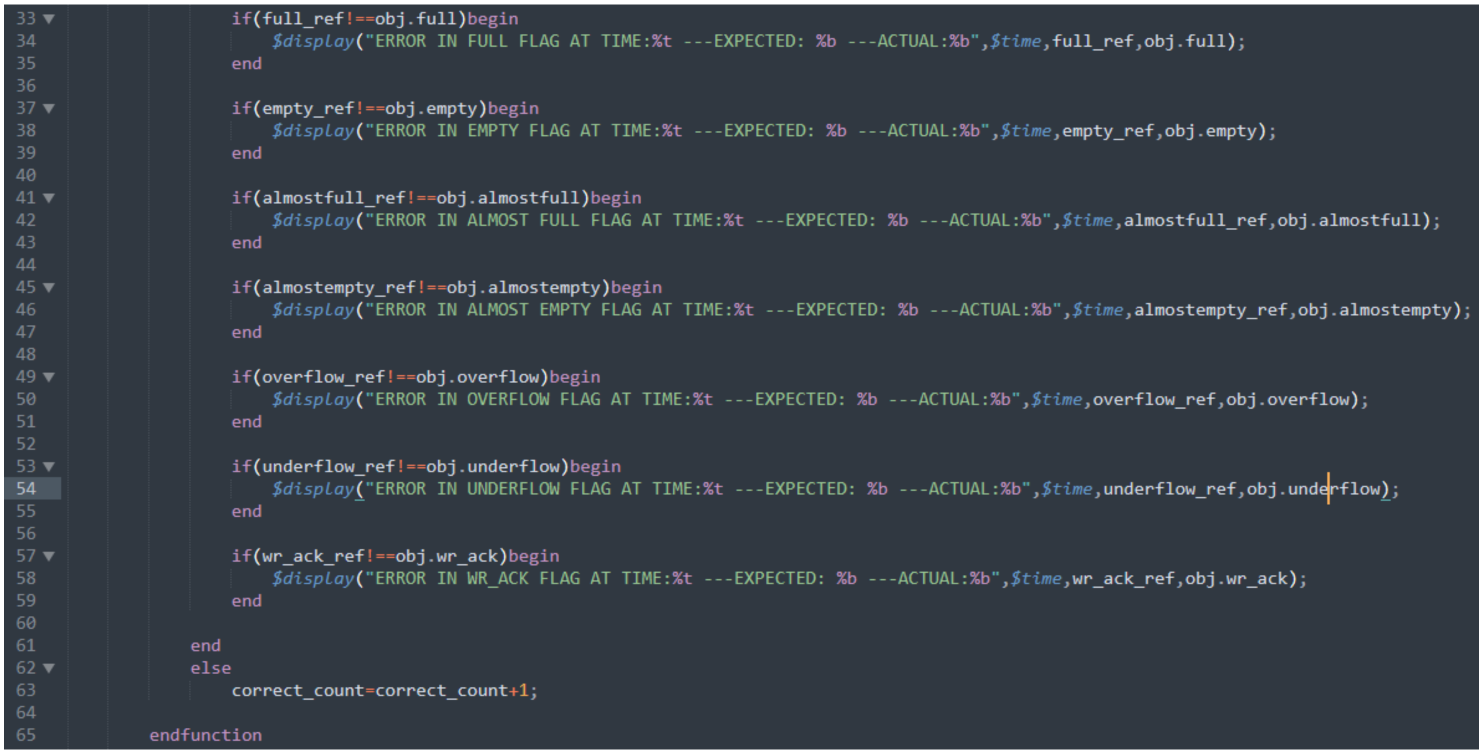


```
            function void sample_data(FIFO_transaction F_txn);

                F_cvg_txn = F_txn   ;

                if(F_cvg_txn.rst_n)
                    mycovergroup.sample();

            endfunction

            function new ();

                mycovergroup = new() ;

            endfunction

        endclass
endpackage
```

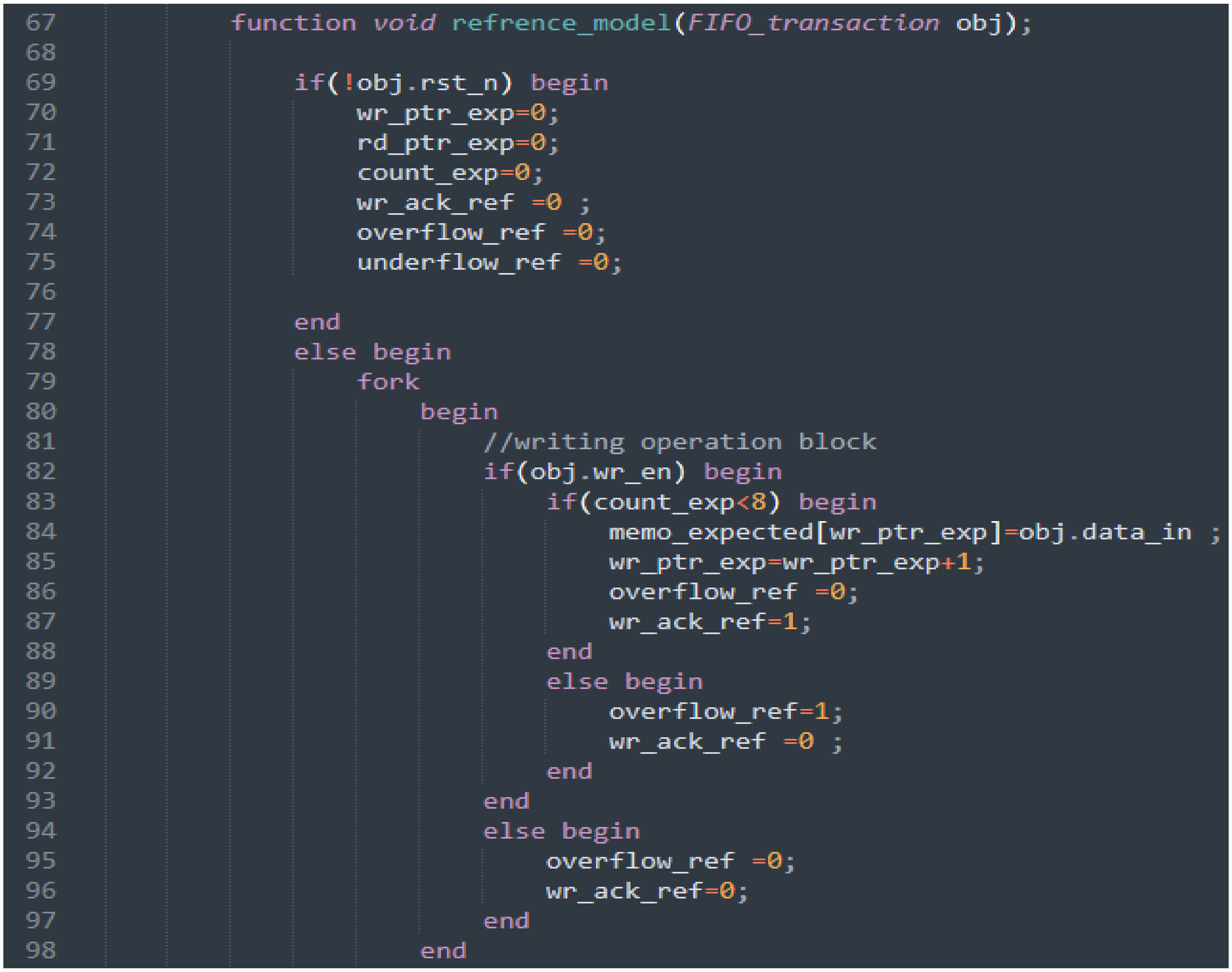
c) Third package:



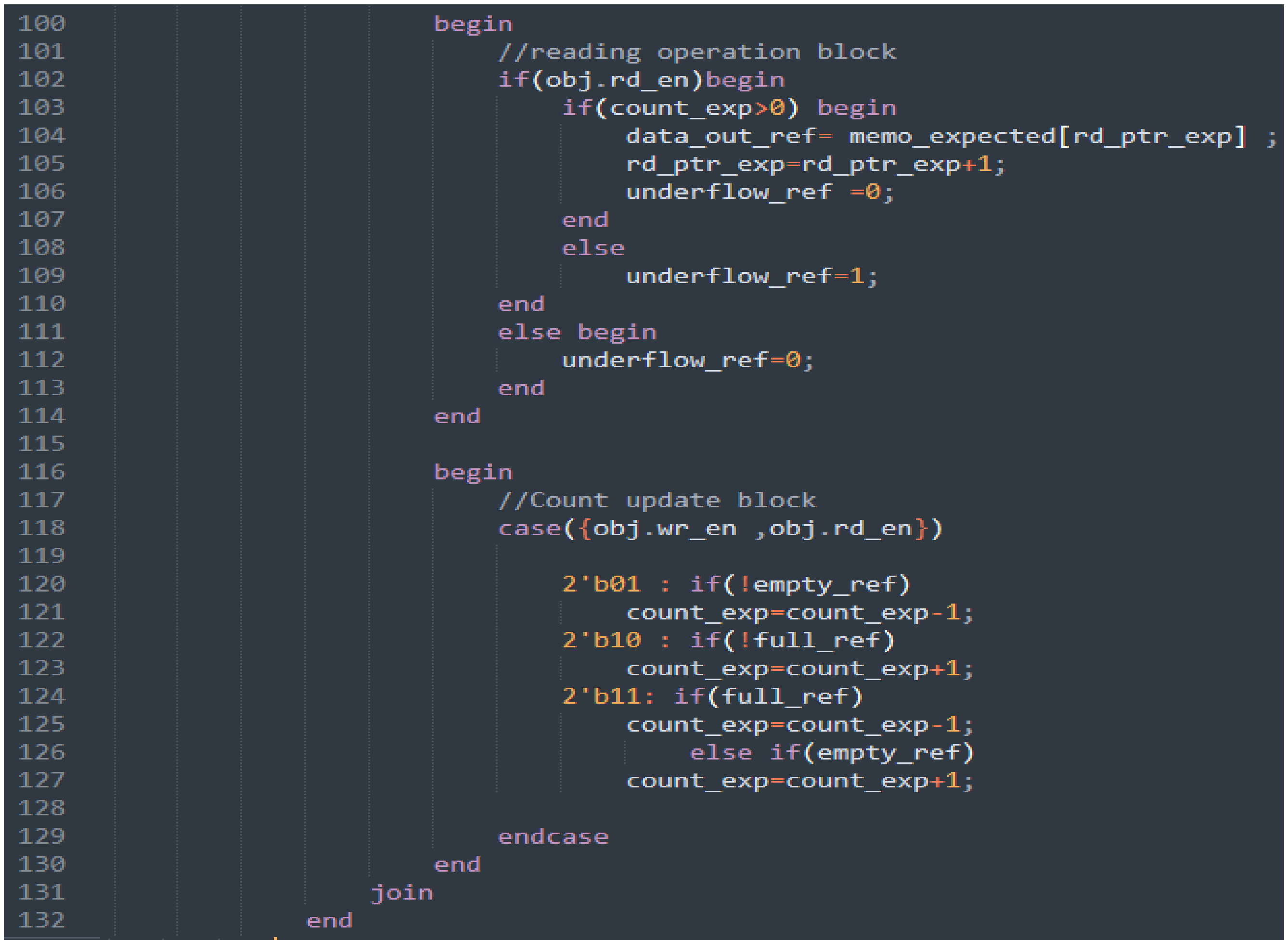
```
package third_pack ;

    import first_pack::*;
    import shared_pkg::*;


    class FIFO_scoreboard;



        logic [FIFO_WIDTH-1:0] data_out_ref;
        logic full_ref,empty_ref,almostfull_ref,almostempty_ref,overflow_ref,underflow_ref,wr_ack_ref ;
        logic [max_fifo_addr-1:0] rd_ptr_exp,wr_ptr_exp;
        logic [max_fifo_addr:0] count_exp ;


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

        function void check_data(FIFO_transaction obj );

            refrence_model(obj);

            if(data_out_ref!== obj.data_out || full_ref!==obj.full || empty_ref!==obj.empty
                || almostfull_ref!==obj.almostfull ||overflow_ref!==obj.overflow
                ||underflow_ref!==obj.underflow ||wr_ack_ref!==obj.wr_ack) begin

                error_count=error_count+1;

                if(data_out_ref!== obj.data_out) begin
                    $display("ERROR IN DATA OUT AT TIME:%t ---EXPECTED: %h ---ACTUAL:%h",$time,data_out_ref,obj.data_out);
                end
```



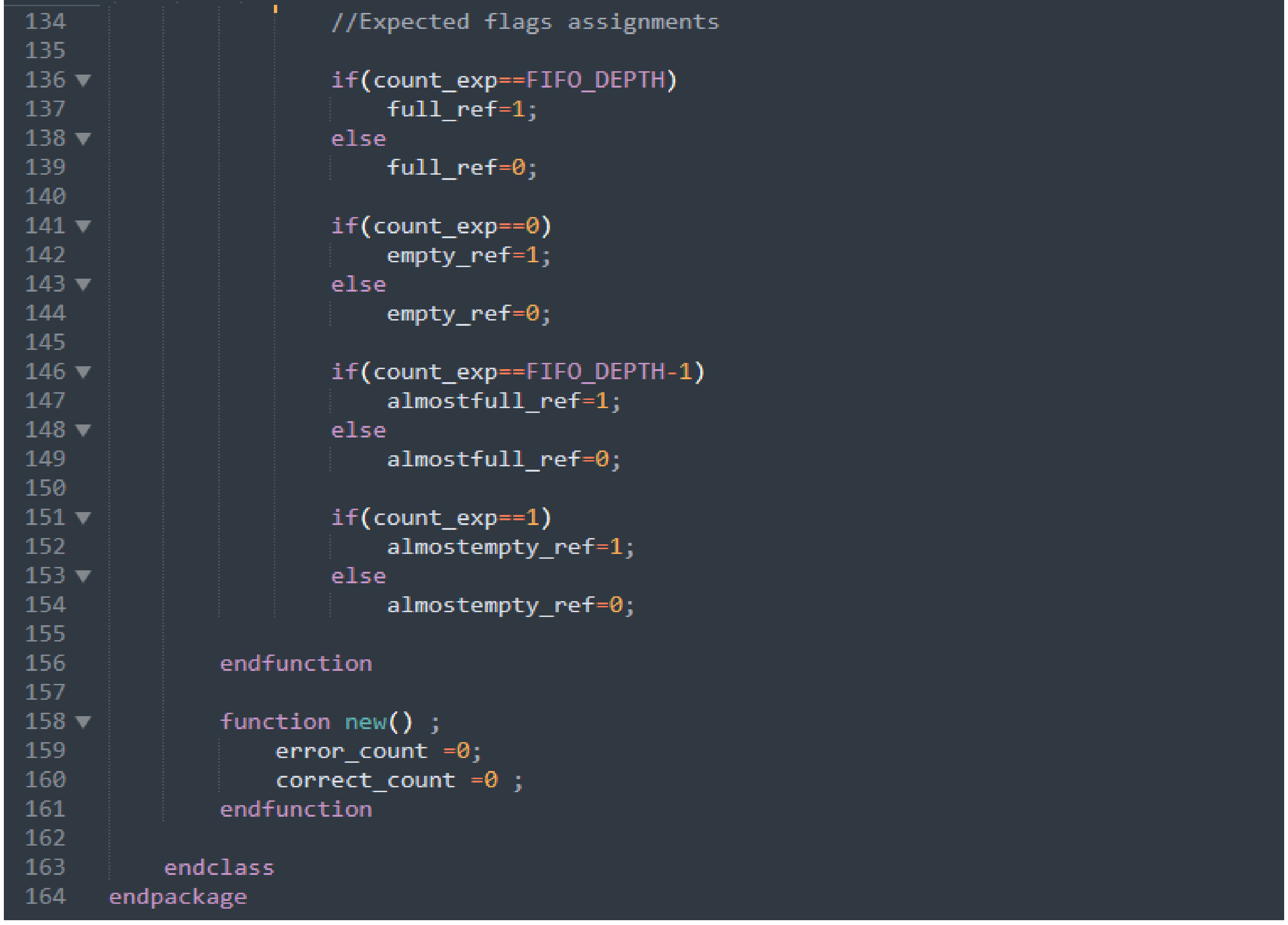
```
            if(full_ref!==obj.full)begin
                $display("ERROR IN FULL FLAG AT TIME:%t ---EXPECTED: %b ---ACTUAL:%b",$time,full_ref,obj.full);
            end

            if(empty_ref!==obj.empty)begin
                $display("ERROR IN EMPTY FLAG AT TIME:%t ---EXPECTED: %b ---ACTUAL:%b",$time,empty_ref,obj.empty);
            end

            if(almostfull_ref!==obj.almostfull)begin
                $display("ERROR IN ALMOST FULL FLAG AT TIME:%t ---EXPECTED: %b ---ACTUAL:%b",$time,almostfull_ref,obj.almostfull);
            end

            if(almostempty_ref!==obj.almostempty)begin
                $display("ERROR IN ALMOST EMPTY FLAG AT TIME:%t ---EXPECTED: %b ---ACTUAL:%b",$time,almostempty_ref,obj.almostempty);
            end

            if(overflow_ref!==obj.overflow)begin
                $display("ERROR IN OVERFLOW FLAG AT TIME:%t ---EXPECTED: %b ---ACTUAL:%b",$time,overflow_ref,obj.overflow);
            end

            if(underflow_ref!==obj.underflow)begin
                $display("ERROR IN UNDERFLOW FLAG AT TIME:%t ---EXPECTED: %b ---ACTUAL:%b",$time,underflow_ref,obj.underflow);
            end

            if(wr_ack_ref!==obj.wr_ack)begin
                $display("ERROR IN WR_ACK FLAG AT TIME:%t ---EXPECTED: %b ---ACTUAL:%b",$time,wr_ack_ref,obj.wr_ack);
            end

        end
        else
            correct_count=correct_count+1;

    endfunction
```



```
        function void refrence_model(FIFO_transaction obj);

            if(!obj.rst_n) begin
                wr_ptr_exp=0;
                rd_ptr_exp=0;
                count_exp=0;
                wr_ack_ref =0 ;
                overflow_ref =0;
                underflow_ref =0;

            end
            else begin
                fork
                    begin
                        //writing operation block
                        if(obj.wr_en) begin
                            if(count_exp<8) begin
                                memo_expected[wr_ptr_exp]=obj.data_in ;
                                wr_ptr_exp=wr_ptr_exp+1;
                                overflow_ref =0;
                                wr_ack_ref=1;
                            end
                            else begin
                                overflow_ref=1;
                                wr_ack_ref =0 ;
                            end
                        end
                        else begin
                            overflow_ref =0;
                            wr_ack_ref=0;
                        end
                    end
```

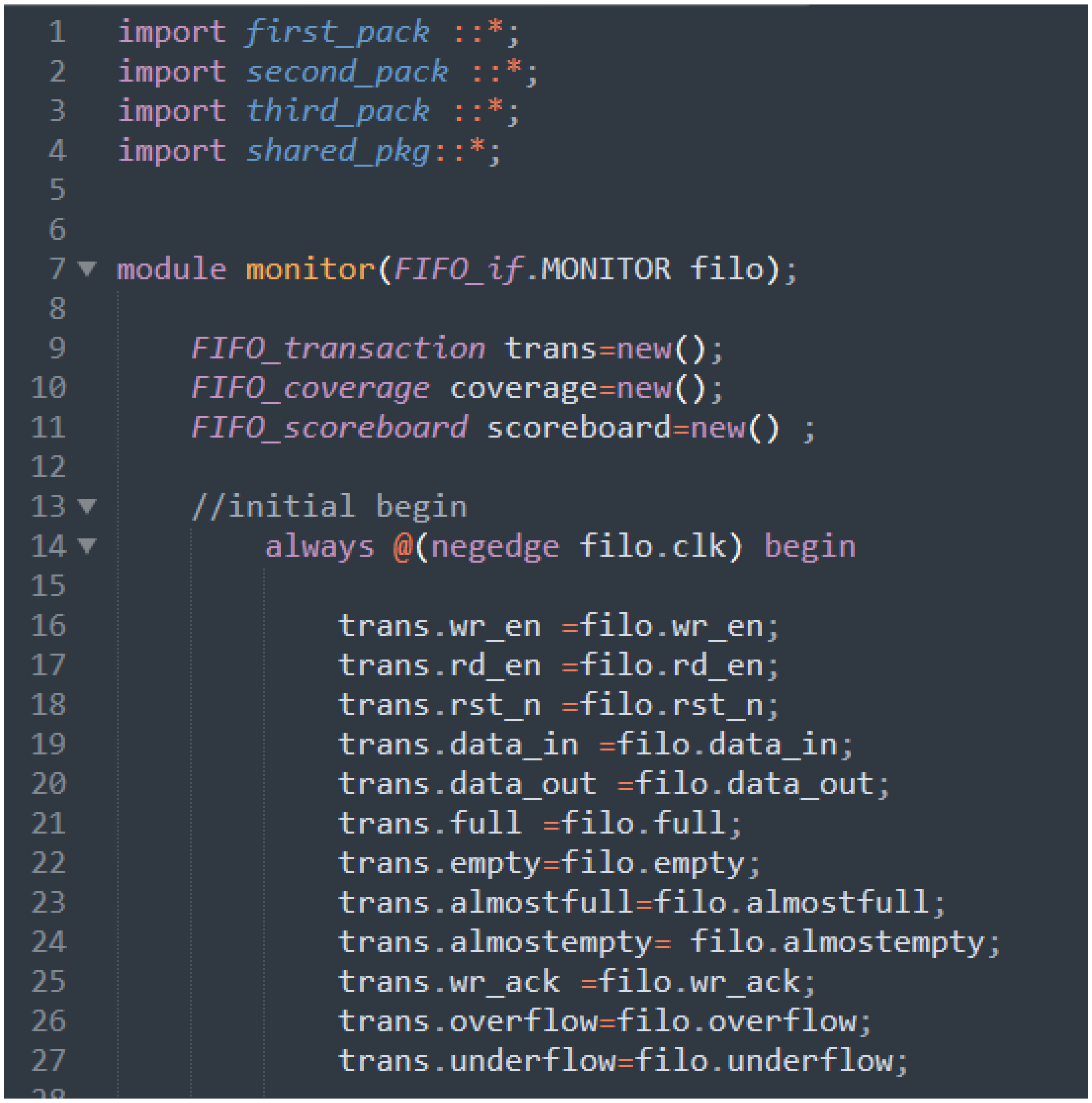


```
                    begin
                        //reading operation block
                        if(obj.rd_en)begin
                            if(count_exp>0) begin
                                data_out_ref= memo_expected[rd_ptr_exp] ;
                                rd_ptr_exp=rd_ptr_exp+1;
                                underflow_ref =0;
                            end
                            else
                                underflow_ref=1;
                        end
                        else begin
                            underflow_ref=0;
                        end
                    end

                    begin
                        //Count update block
                        case({obj.wr_en ,obj.rd_en})

                            2'b01 : if(!empty_ref)
                                count_exp=count_exp-1;
                            2'b10 : if(!full_ref)
                                count_exp=count_exp+1;
                            2'b11: if(full_ref)
                                count_exp=count_exp-1;
                                    else if(empty_ref)
                                count_exp=count_exp+1;

                        endcase
                    end
                join
            end
```

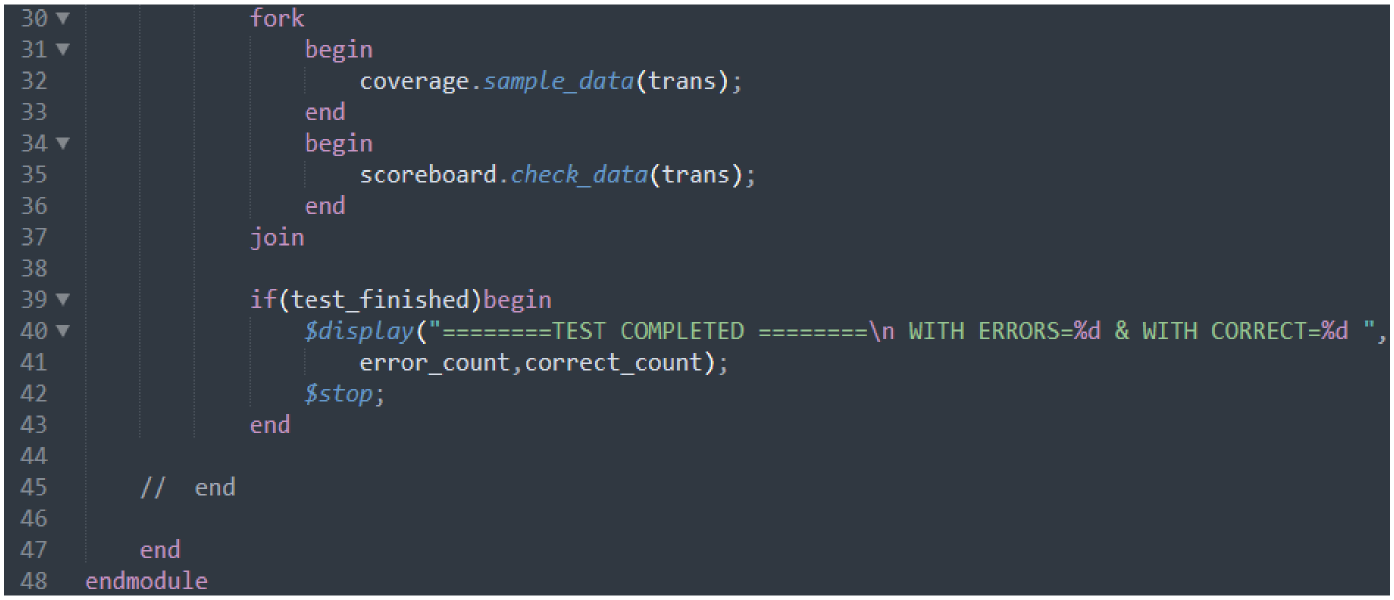


```
                //Expected flags assignments

                if(count_exp==FIFO_DEPTH)
                    full_ref=1;
                else
                    full_ref=0;

                if(count_exp==0)
                    empty_ref=1;
                else
                    empty_ref=0;

                if(count_exp==FIFO_DEPTH-1)
                    almostfull_ref=1;
                else
                    almostfull_ref=0;

                if(count_exp==1)
                    almostempty_ref=1;
                else
                    almostempty_ref=0;

        endfunction

        function new() ;
            error_count =0;
            correct_count =0 ;
        endfunction

    endclass
endpackage
```

d) Monitor module:

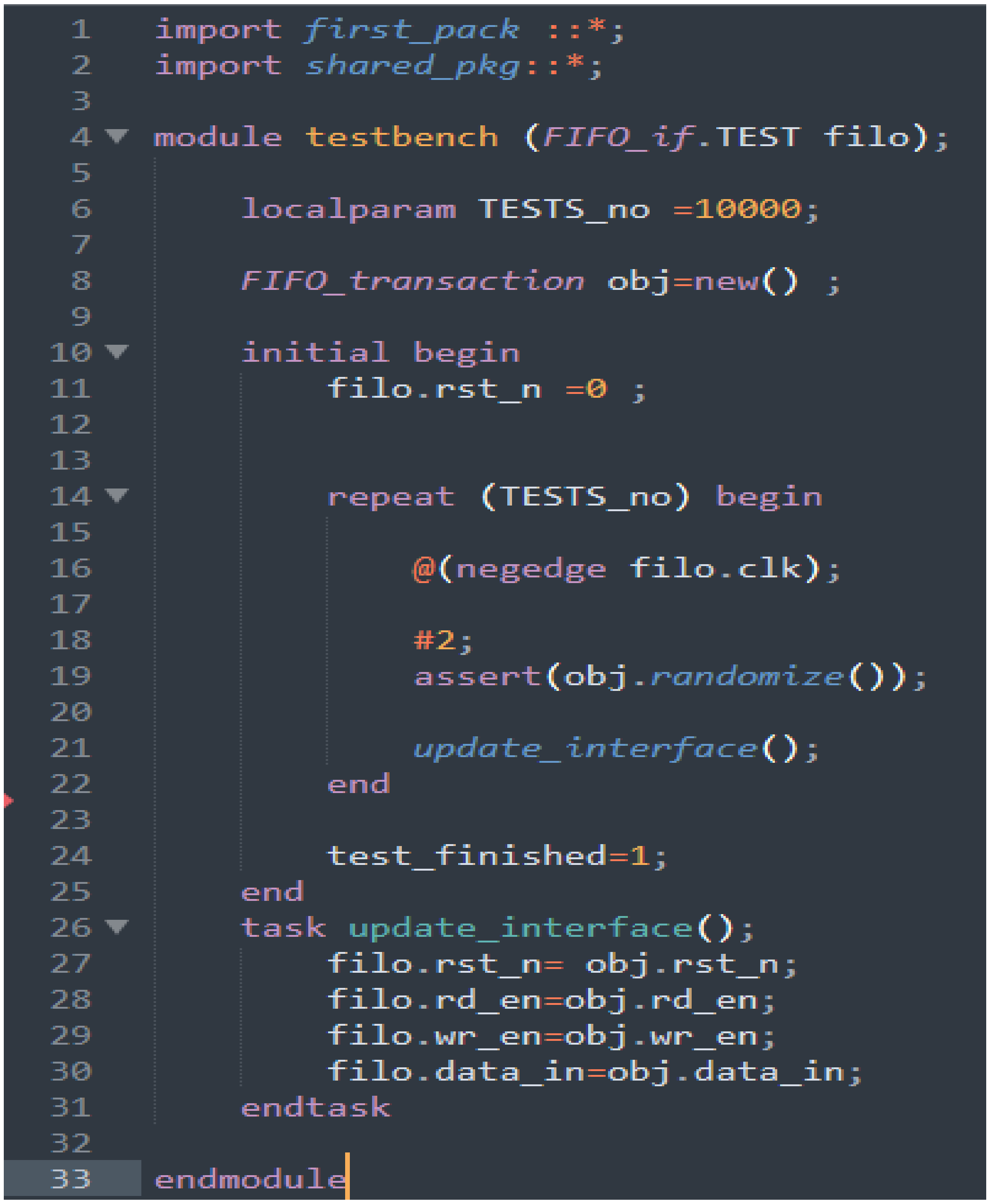


```
import first_pack ::*;
import second_pack ::*;
import third_pack ::*;
import shared_pkg::*;


module monitor(FIFO_if.MONITOR filo);

    FIFO_transaction trans=new();
    FIFO_coverage coverage=new();
    FIFO_scoreboard scoreboard=new() ;

    //initial begin
        always @(negedge filo.clk) begin

            trans.wr_en =filo.wr_en;
            trans.rd_en =filo.rd_en;
            trans.rst_n =filo.rst_n;
            trans.data_in =filo.data_in;
            trans.data_out =filo.data_out;
            trans.full =filo.full;
            trans.empty=filo.empty;
            trans.almostfull=filo.almostfull;
            trans.almostempty= filo.almostempty;
            trans.wr_ack =filo.wr_ack;
            trans.overflow=filo.overflow;
            trans.underflow=filo.underflow;
```



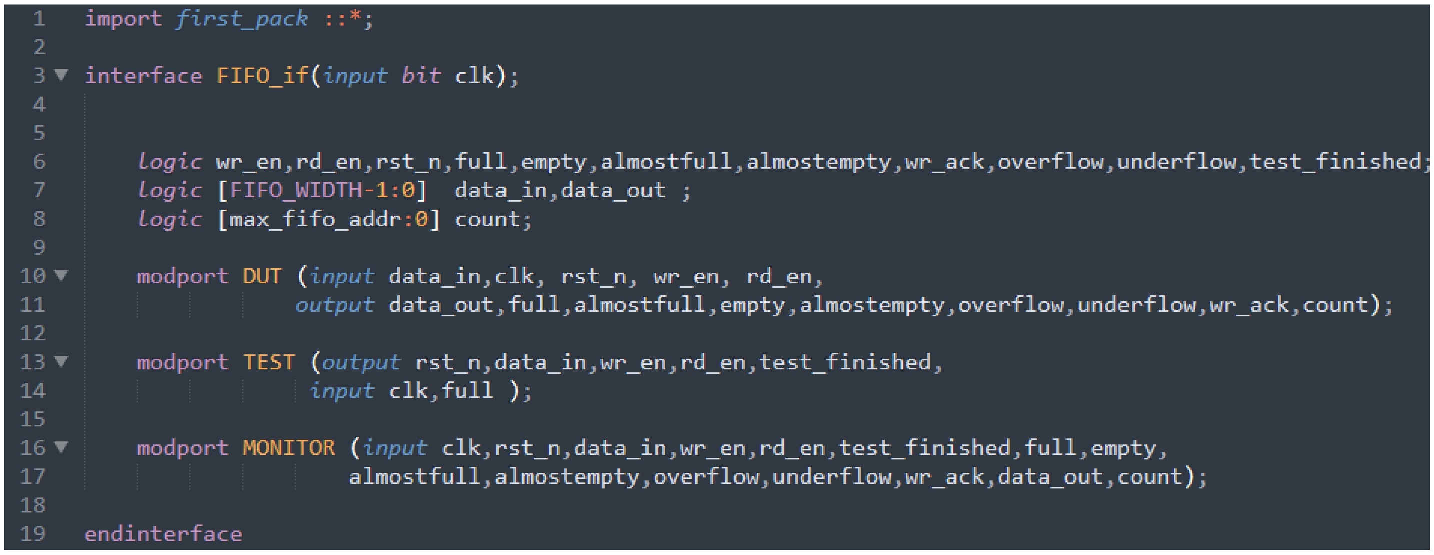
```
                fork
                    begin
                        coverage.sample_data(trans);
                    end
                    begin
                        scoreboard.check_data(trans);
                    end
                join

                if(test_finished)begin
                    $display("=======TEST COMPLETED ========\n WITH ERRORS=%d & WITH CORRECT=%d ",
                        error_count,correct_count);
                    $stop;
                end

        //  end

        end
endmodule
```

e) Testbench:



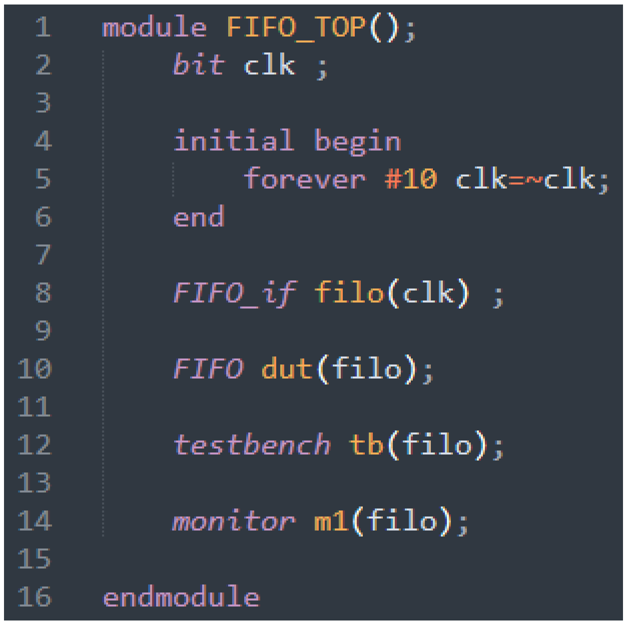
```
import first_pack ::*;
import shared_pkg::*;

module testbench (FIFO_if.TEST filo);

    localparam TESTS_no =10000;

    FIFO_transaction obj=new() ;

    initial begin
        filo.rst_n =0 ;


        repeat (TESTS_no) begin

            @(negedge filo.clk);

            #2;
            assert(obj.randomize());

            update_interface();
        end

        test_finished=1;
    end
    task update_interface();
        filo.rst_n= obj.rst_n;
        filo.rd_en=obj.rd_en;
        filo.wr_en=obj.wr_en;
        filo.data_in=obj.data_in;
    endtask

endmodule
```

f) Interface:



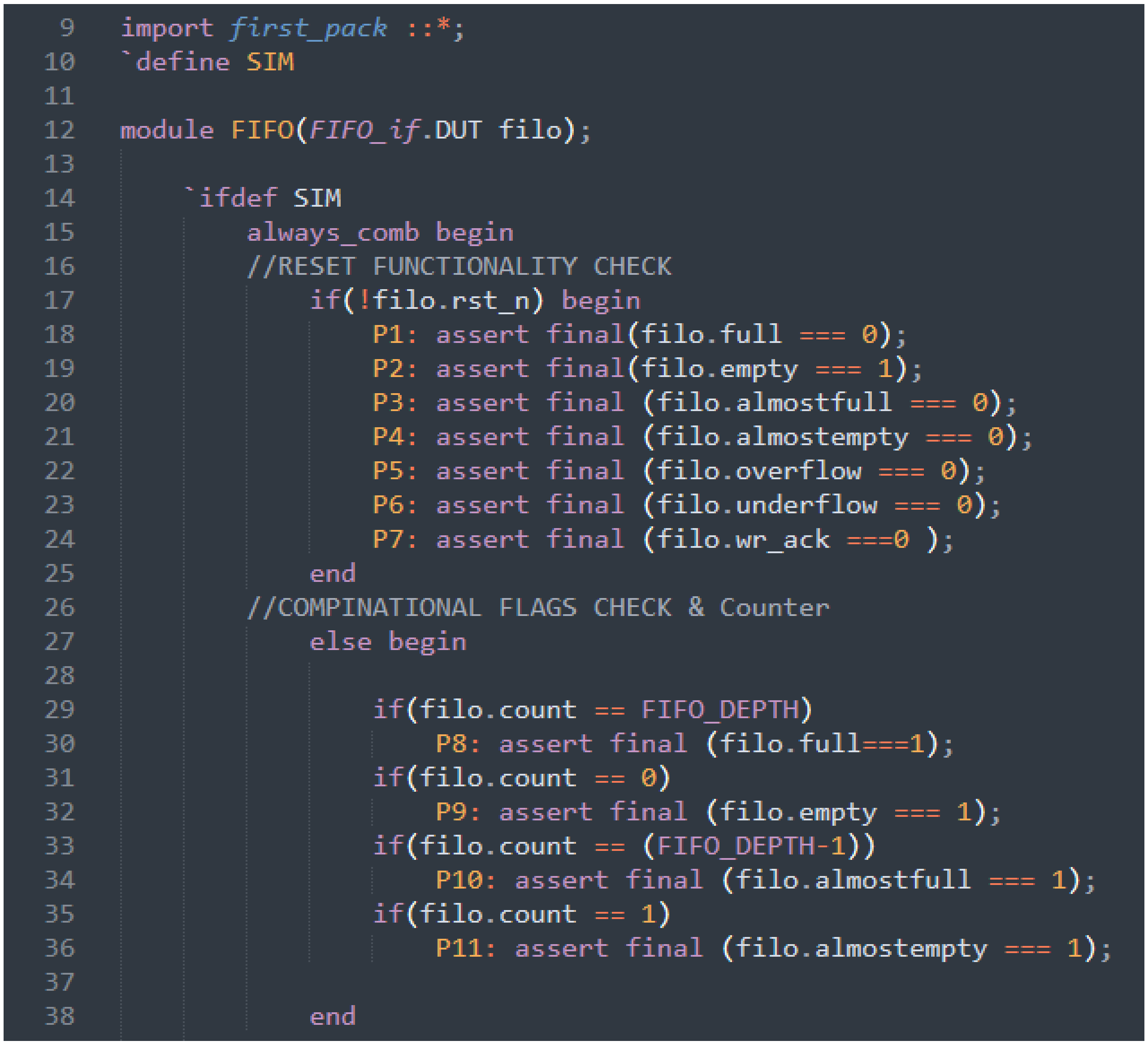
```
import first_pack ::*;

interface FIFO_if(input bit clk);


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

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

    modport TEST (output rst_n,data_in,wr_en,rd_en,test_finished,
                  input clk,full );

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

endinterface
```

g) Top module:

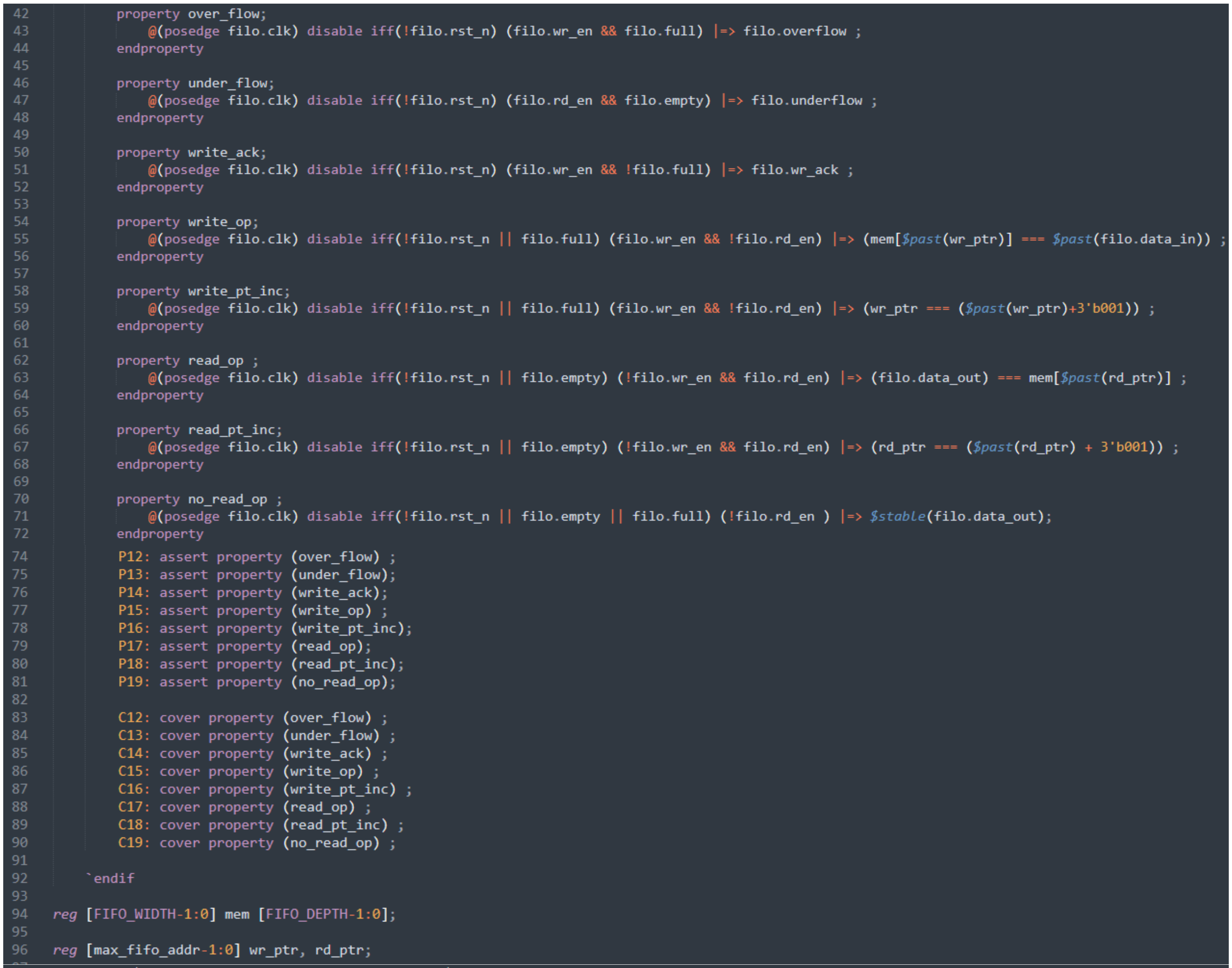


```
module FIFO_TOP();
    bit clk ;

    initial begin
        forever #10 clk=~clk;
    end

    FIFO_if filo(clk) ;

    FIFO dut(filo);

    testbench tb(filo);

    monitor m1(filo);

endmodule
```

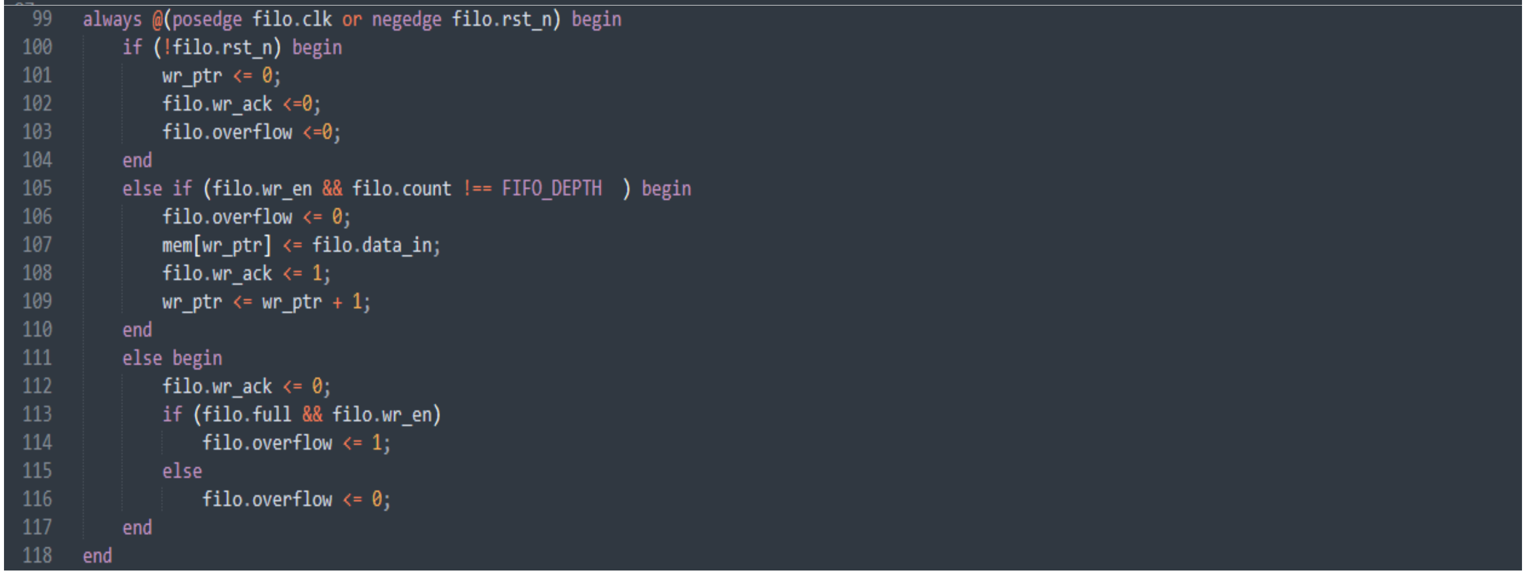
h) New Design RTL:



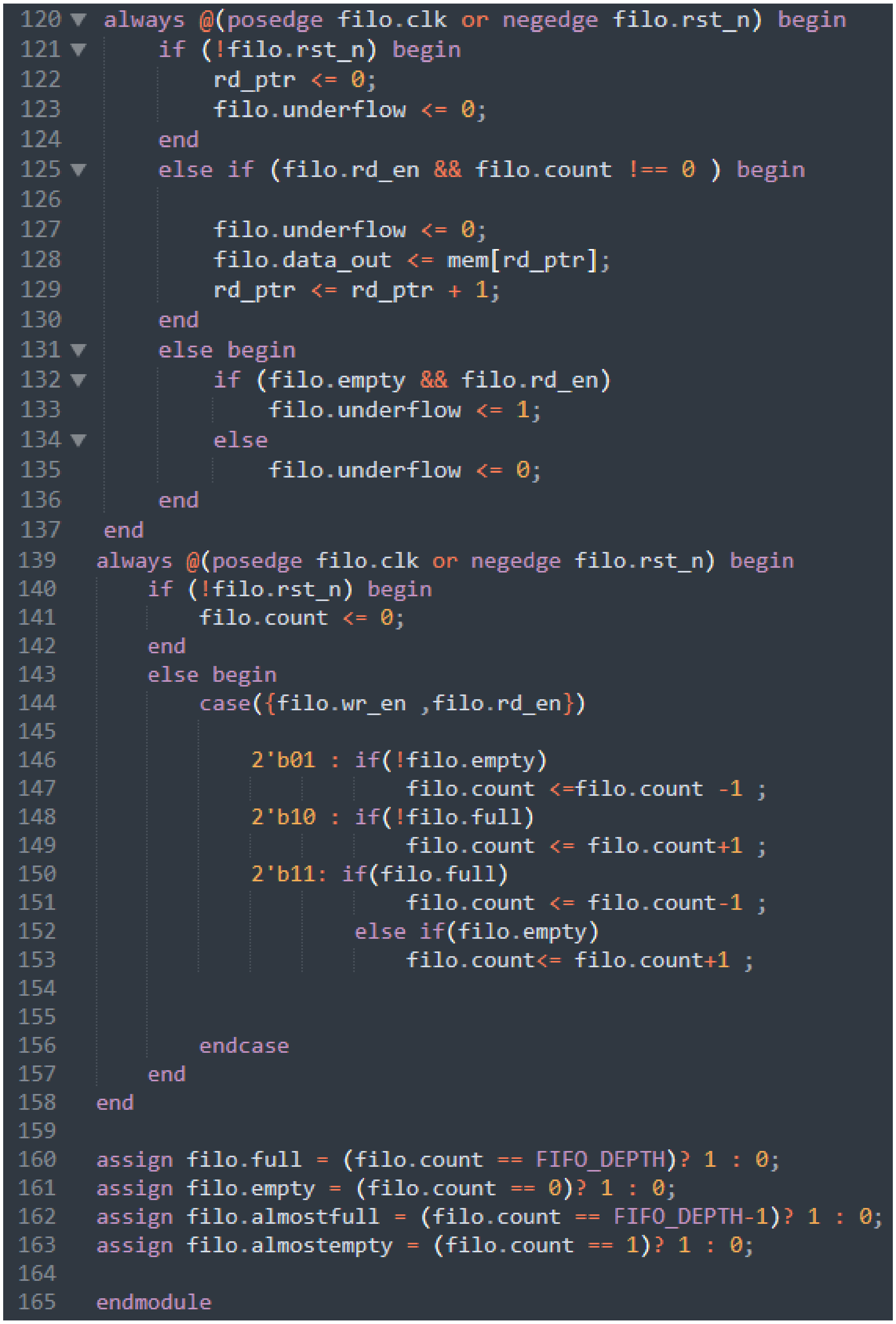
```
import first_pack ::*;
`define SIM

module FIFO(FIFO_if.DUT filo);

    `ifdef SIM
        always_comb begin
        //RESET FUNCTIONALITY CHECK
            if(!filo.rst_n) begin
                P1: assert final(filo.full === 0);
                P2: assert final(filo.empty === 1);
                P3: assert final (filo.almostfull === 0);
                P4: assert final (filo.almostempty === 0);
                P5: assert final (filo.overflow === 0);
                P6: assert final (filo.underflow === 0);
                P7: assert final (filo.wr_ack ===0 );
            end
        //COMPINATIONAL FLAGS CHECK & Counter
            else begin

                if(filo.count == FIFO_DEPTH)
                    P8: assert final (filo.full===1);
                if(filo.count == 0)
                    P9: assert final (filo.empty === 1);
                if(filo.count == (FIFO_DEPTH-1))
                    P10: assert final (filo.almostfull === 1);
                if(filo.count == 1)
                    P11: assert final (filo.almostempty === 1);

            end
```



```systemverilog
        property over_flow;
            @(posedge filo.clk) disable iff(!filo.rst_n) (filo.wr_en && filo.full) |=> filo.overflow ;
        endproperty

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

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

        property write_op;
            @(posedge filo.clk) disable iff(!filo.rst_n || filo.full) (filo.wr_en && !filo.rd_en) |=> (mem[$past(wr_ptr)] === $past(filo.data_in)) ;
        endproperty

        property write_pt_inc;
            @(posedge filo.clk) disable iff(!filo.rst_n || filo.full) (filo.wr_en && !filo.rd_en) |=> (wr_ptr === ($past(wr_ptr)+3'b001)) ;
        endproperty

        property read_op ;
            @(posedge filo.clk) disable iff(!filo.rst_n || filo.empty) (!filo.wr_en && filo.rd_en) |=> (filo.data_out) === mem[$past(rd_ptr)] ;
        endproperty

        property read_pt_inc;
            @(posedge filo.clk) disable iff(!filo.rst_n || filo.empty) (!filo.wr_en && filo.rd_en) |=> (rd_ptr === ($past(rd_ptr) + 3'b001)) ;
        endproperty

        property no_read_op ;
            @(posedge filo.clk) disable iff(!filo.rst_n || filo.empty || filo.full) (!filo.rd_en ) |=> $stable(filo.data_out);
        endproperty
        P12: assert property (over_flow) ;
        P13: assert property (under_flow);
        P14: assert property (write_ack);
        P15: assert property (write_op) ;
        P16: assert property (write_pt_inc);
        P17: assert property (read_op);
        P18: assert property (read_pt_inc);
        P19: assert property (no_read_op);

        C12: cover property (over_flow) ;
        C13: cover property (under_flow) ;
        C14: cover property (write_ack) ;
        C15: cover property (write_op) ;
        C16: cover property (write_pt_inc) ;
        C17: cover property (read_op) ;
        C18: cover property (read_pt_inc) ;
        C19: cover property (no_read_op) ;

    `endif

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

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



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



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

        filo.underflow <= 0;
        filo.data_out <= mem[rd_ptr];
        rd_ptr <= rd_ptr + 1;
    end
    else begin
        if (filo.empty && filo.rd_en)
            filo.underflow <= 1;
        else
            filo.underflow <= 0;
    end
end
always @(posedge filo.clk or negedge filo.rst_n) begin
    if (!filo.rst_n) begin
        filo.count <= 0;
    end
    else begin
        case({filo.wr_en ,filo.rd_en})

            2'b01 : if(!filo.empty)
                        filo.count <=filo.count -1 ;
            2'b10 : if(!filo.full)
                        filo.count <= filo.count+1 ;
            2'b11: if(filo.full)
                        filo.count <= filo.count-1 ;
                    else if(filo.empty)
                        filo.count<= filo.count+1 ;


        endcase
    end
end

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

endmodule
```