# Mohamed_Hatem_Abdelmenem_FIFO.sv

## Verification plan:

1- inlization the fifo

2- Randmize the input  and pass it to obj in transaction class

3-check data function and sample function

4 cereat refrence task

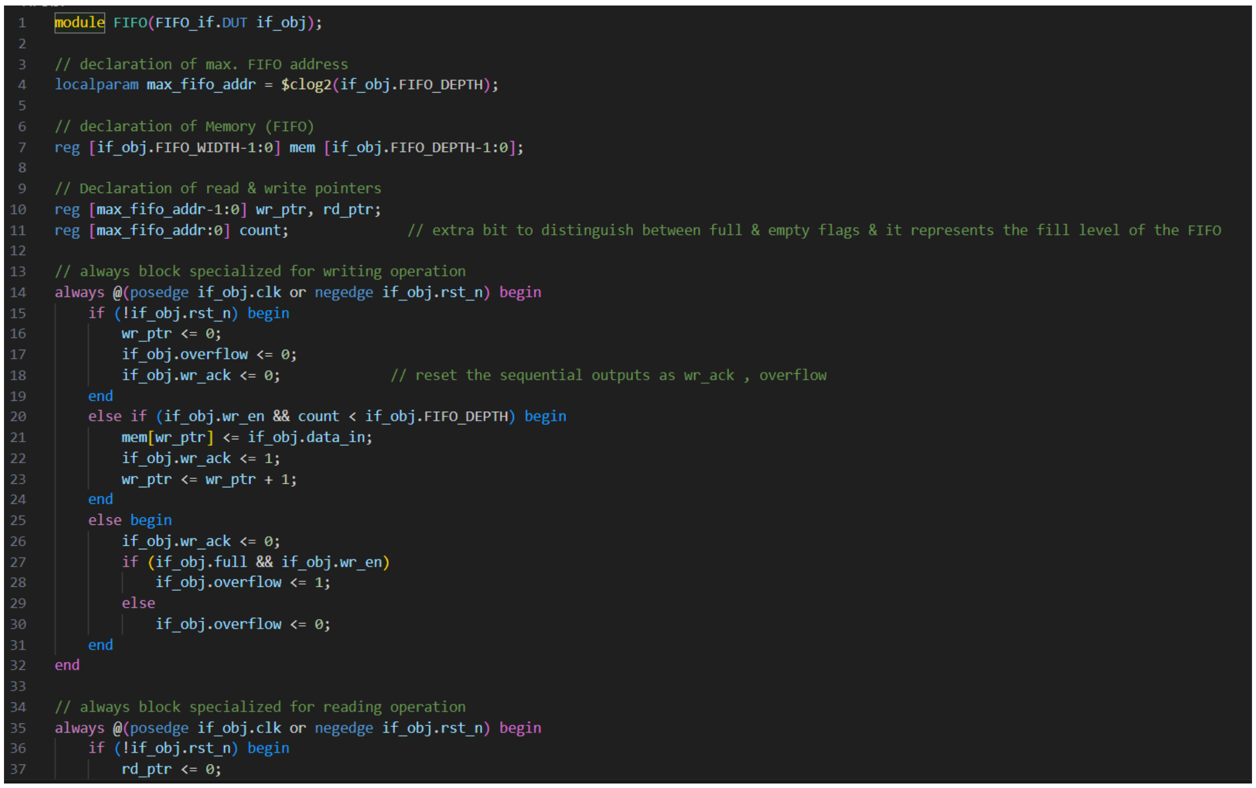
5-count will increment when data is correct and decrement when data is not correct

6- stop when test_finished =1

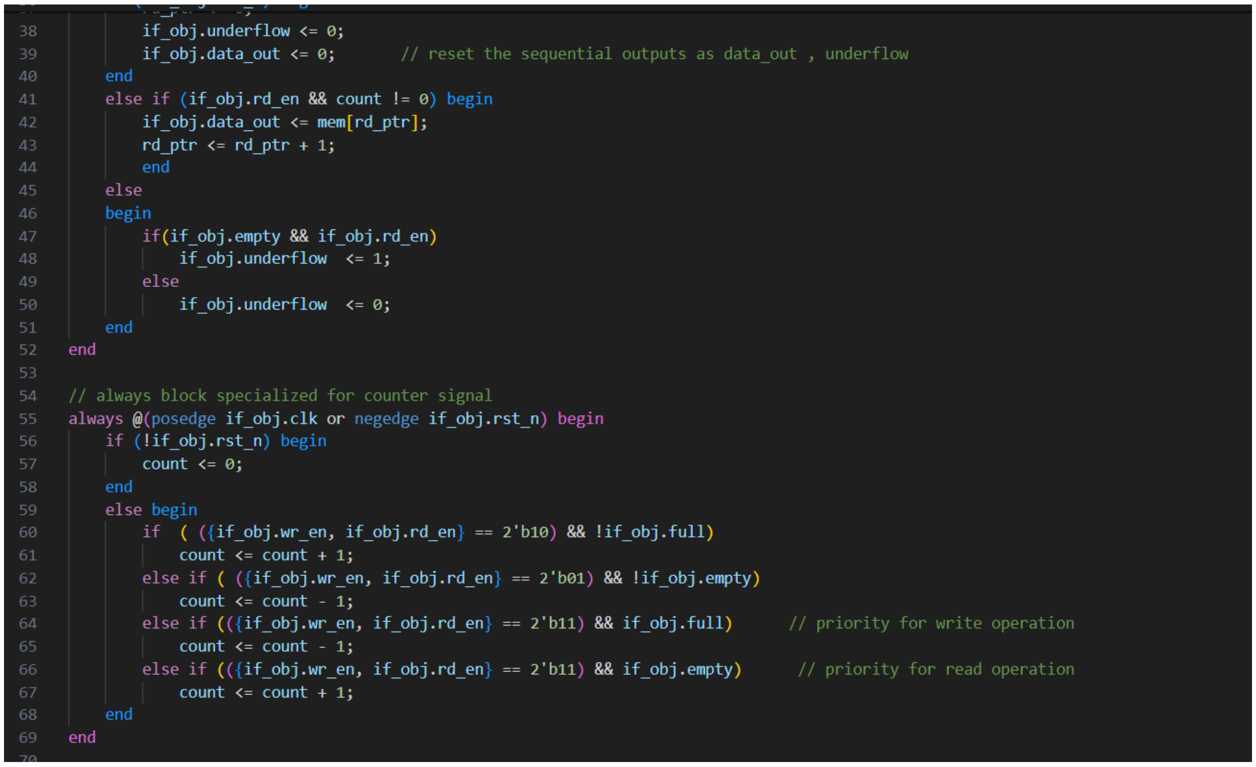
List of Bugs in RTL

1- Underflow is combinational
2- Almost full is (FIFODEPTH -2)
3- In Reset Operation
4- Cases of counter if rd_rn & er_en are both active &  fifo is full so the priority  will be for read operation is not covered
5- Cases of counter if rd_rn & er_en are both active &  fifo is full so the priority  will be for write operation is not covered

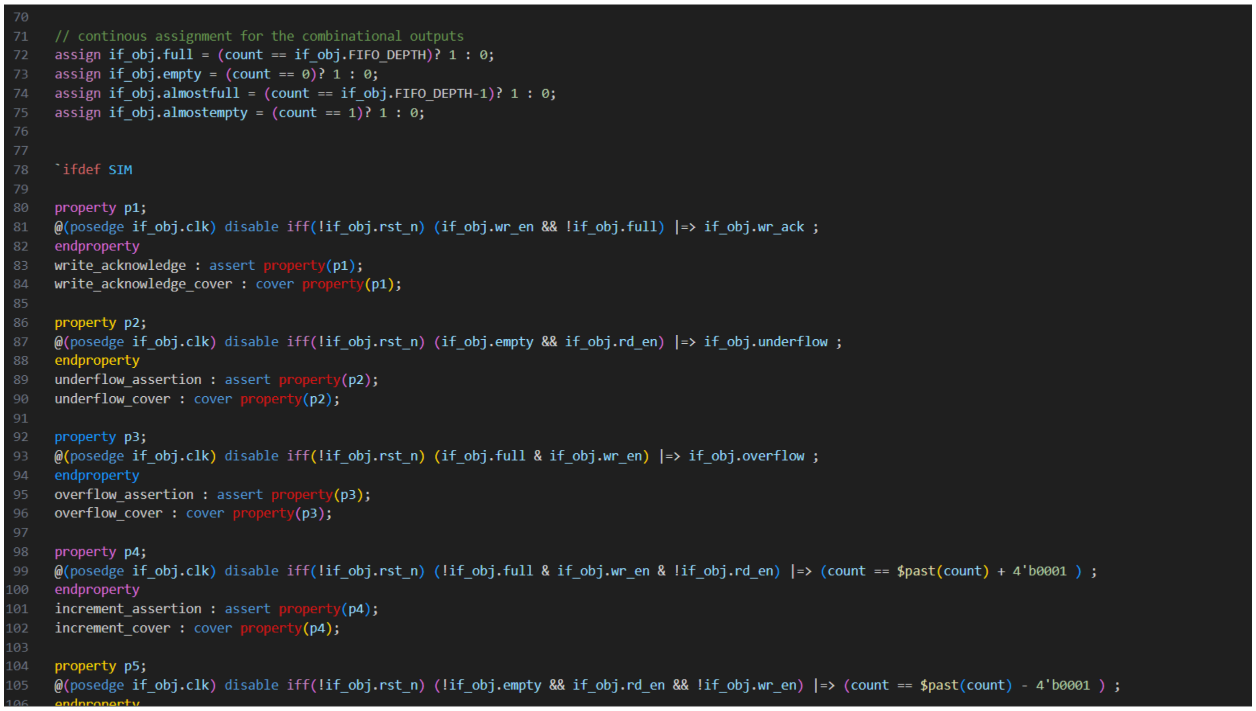
RTL after fixed bugs :



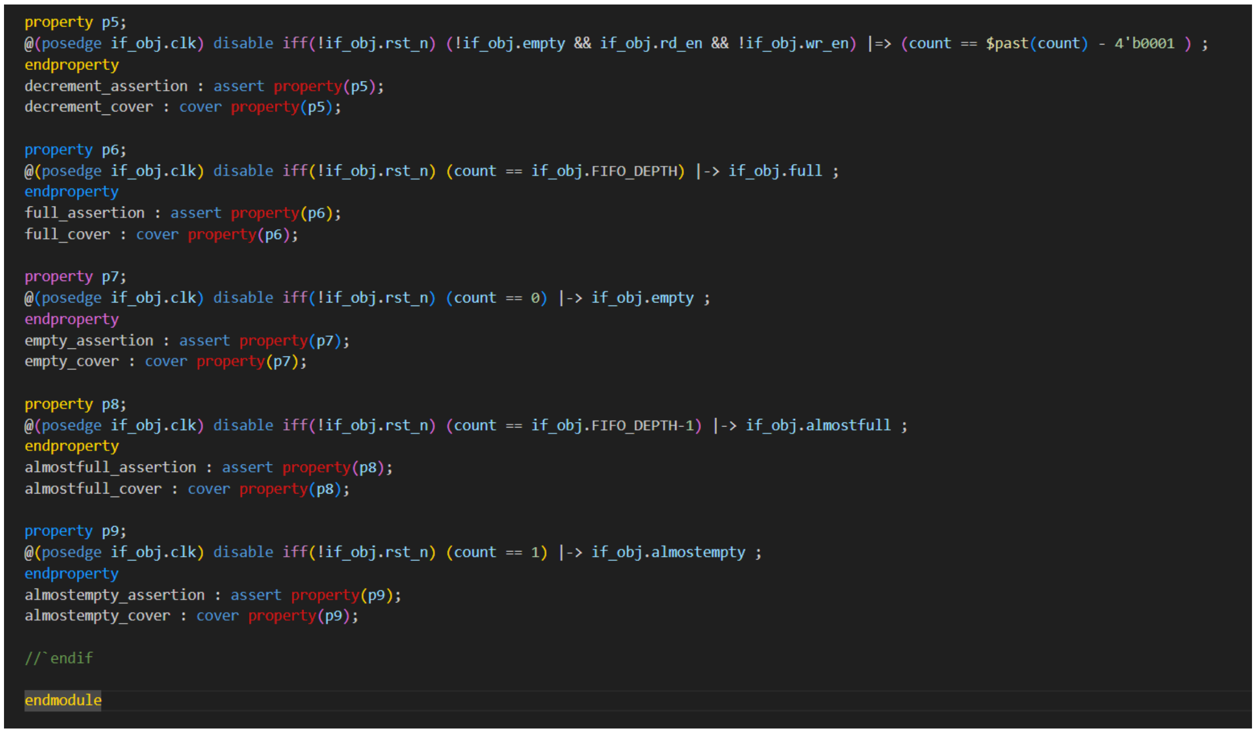
```
module FIFO(FIFO_if.DUT if_obj);

// declaration of max. FIFO address
localparam max_fifo_addr = $clog2(if_obj.FIFO_DEPTH);

// declaration of Memory (FIFO)
reg [if_obj.FIFO_WIDTH-1:0] mem [if_obj.FIFO_DEPTH-1:0];

// Declaration of read & write pointers
reg [max_fifo_addr-1:0] wr_ptr, rd_ptr;
reg [max_fifo_addr:0] count;               // extra bit to distinguish between full & empty flags & it represents the fill level of the FIFO

// always block specialized for writing operation
always @(posedge if_obj.clk or negedge if_obj.rst_n) begin
    if (!if_obj.rst_n) begin
        wr_ptr <= 0;
        if_obj.overflow <= 0;
        if_obj.wr_ack <= 0;             // reset the sequential outputs as wr_ack , overflow
    end
    else if (if_obj.wr_en && count < if_obj.FIFO_DEPTH) begin
        mem[wr_ptr] <= if_obj.data_in;
        if_obj.wr_ack <= 1;
        wr_ptr <= wr_ptr + 1;
    end
    else begin
        if_obj.wr_ack <= 0;
        if (if_obj.full && if_obj.wr_en)
            if_obj.overflow <= 1;
        else
            if_obj.overflow <= 0;
    end
end

// always block specialized for reading operation
always @(posedge if_obj.clk or negedge if_obj.rst_n) begin
    if (!if_obj.rst_n) begin
        rd_ptr <= 0;
```



```
        if_obj.underflow <= 0;
        if_obj.data_out <= 0;       // reset the sequential outputs as data_out , underflow
    end
    else if (if_obj.rd_en && count != 0) begin
        if_obj.data_out <= mem[rd_ptr];
        rd_ptr <= rd_ptr + 1;
        end
    else
    begin
        if(if_obj.empty && if_obj.rd_en)
            if_obj.underflow  <= 1;
        else
            if_obj.underflow  <= 0;
    end
end

// always block specialized for counter signal
always @(posedge if_obj.clk or negedge if_obj.rst_n) begin
    if (!if_obj.rst_n) begin
        count <= 0;
    end
    else begin
        if  ( ({if_obj.wr_en, if_obj.rd_en} == 2'b10) && !if_obj.full)
            count <= count + 1;
        else if ( ({if_obj.wr_en, if_obj.rd_en} == 2'b01) && !if_obj.empty)
            count <= count - 1;
        else if (({if_obj.wr_en, if_obj.rd_en} == 2'b11) && if_obj.full)      // priority for write operation
            count <= count - 1;
        else if (({if_obj.wr_en, if_obj.rd_en} == 2'b11) && if_obj.empty)      // priority for read operation
            count <= count + 1;
    end
end
```



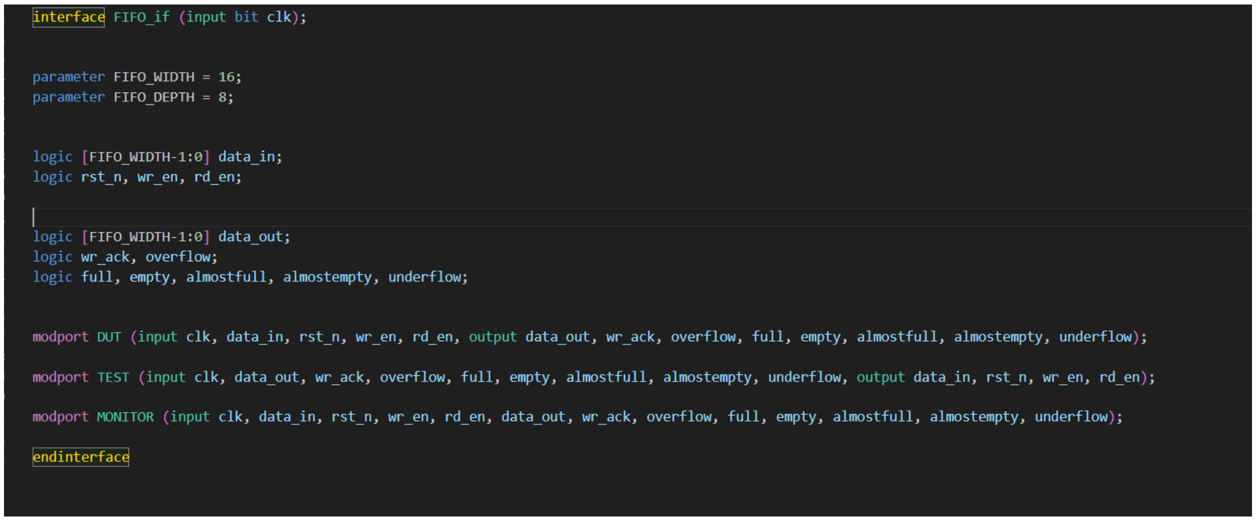
```
// continous assignment for the combinational outputs
assign if_obj.full = (count == if_obj.FIFO_DEPTH)? 1 : 0;
assign if_obj.empty = (count == 0)? 1 : 0;
assign if_obj.almostfull = (count == if_obj.FIFO_DEPTH-1)? 1 : 0;
assign if_obj.almostempty = (count == 1)? 1 : 0;


`ifdef SIM

property p1;
@(posedge if_obj.clk) disable iff(!if_obj.rst_n) (if_obj.wr_en && !if_obj.full) |=> if_obj.wr_ack ;
endproperty
write_acknowledge : assert property(p1);
write_acknowledge_cover : cover property(p1);

property p2;
@(posedge if_obj.clk) disable iff(!if_obj.rst_n) (if_obj.empty && if_obj.rd_en) |=> if_obj.underflow ;
endproperty
underflow_assertion : assert property(p2);
underflow_cover : cover property(p2);

property p3;
@(posedge if_obj.clk) disable iff(!if_obj.rst_n) (if_obj.full & if_obj.wr_en) |=> if_obj.overflow ;
endproperty
overflow_assertion : assert property(p3);
overflow_cover : cover property(p3);

property p4;
@(posedge if_obj.clk) disable iff(!if_obj.rst_n) (!if_obj.full & if_obj.wr_en & !if_obj.rd_en) |=> (count == $past(count) + 4'b0001 ) ;
endproperty
increment_assertion : assert property(p4);
increment_cover : cover property(p4);

property p5;
@(posedge if_obj.clk) disable iff(!if_obj.rst_n) (!if_obj.empty && if_obj.rd_en && !if_obj.wr_en) |=> (count == $past(count) - 4'b0001 ) ;
endproperty
```



```
property p5;
@(posedge if_obj.clk) disable iff(!if_obj.rst_n) (!if_obj.empty && if_obj.rd_en && !if_obj.wr_en) |=> (count == $past(count) - 4'b0001 ) ;
endproperty
```

```
decrement_assertion : assert property(p5);
decrement_cover : cover property(p5);

property p6;
@(posedge if_obj.clk) disable iff(!if_obj.rst_n) (count == if_obj.FIFO_DEPTH) |-> if_obj.full ;
endproperty
full_assertion : assert property(p6);
full_cover : cover property(p6);

property p7;
@(posedge if_obj.clk) disable iff(!if_obj.rst_n) (count == 0) |-> if_obj.empty ;
endproperty
empty_assertion : assert property(p7);
empty_cover : cover property(p7);

property p8;
@(posedge if_obj.clk) disable iff(!if_obj.rst_n) (count == if_obj.FIFO_DEPTH-1) |-> if_obj.almostfull ;
endproperty
almostfull_assertion : assert property(p8);
almostfull_cover : cover property(p8);

property p9;
@(posedge if_obj.clk) disable iff(!if_obj.rst_n) (count == 1) |-> if_obj.almostempty ;
endproperty
almostempty_assertion : assert property(p9);
almostempty_cover : cover property(p9);

//`endif

endmodule
```

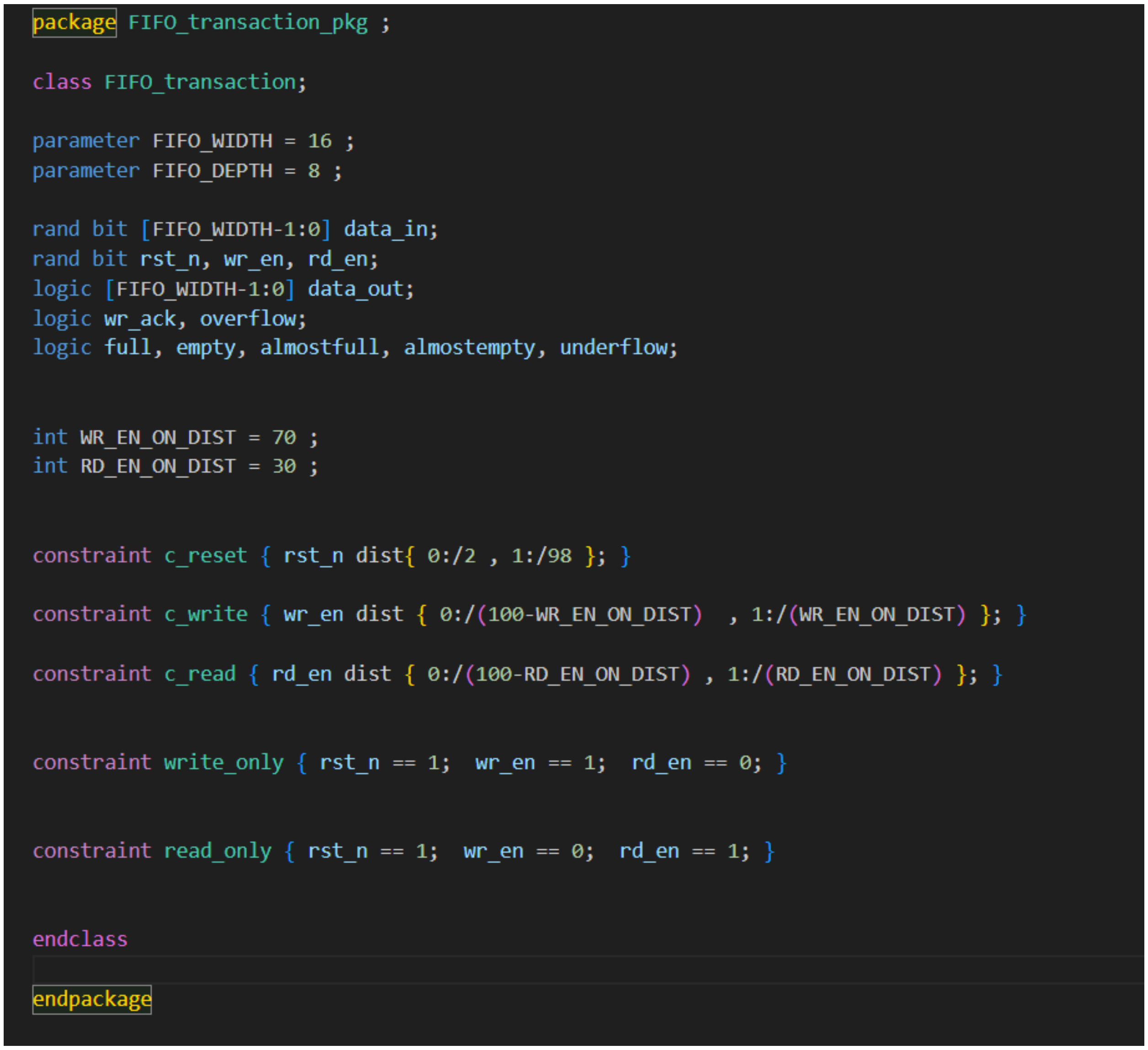
## Interface:



```
interface FIFO_if (input bit 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 clk, data_in, rst_n, wr_en, rd_en, output data_out, wr_ack, overflow, full, empty, almostfull, almostempty, underflow);

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

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

endinterface
```

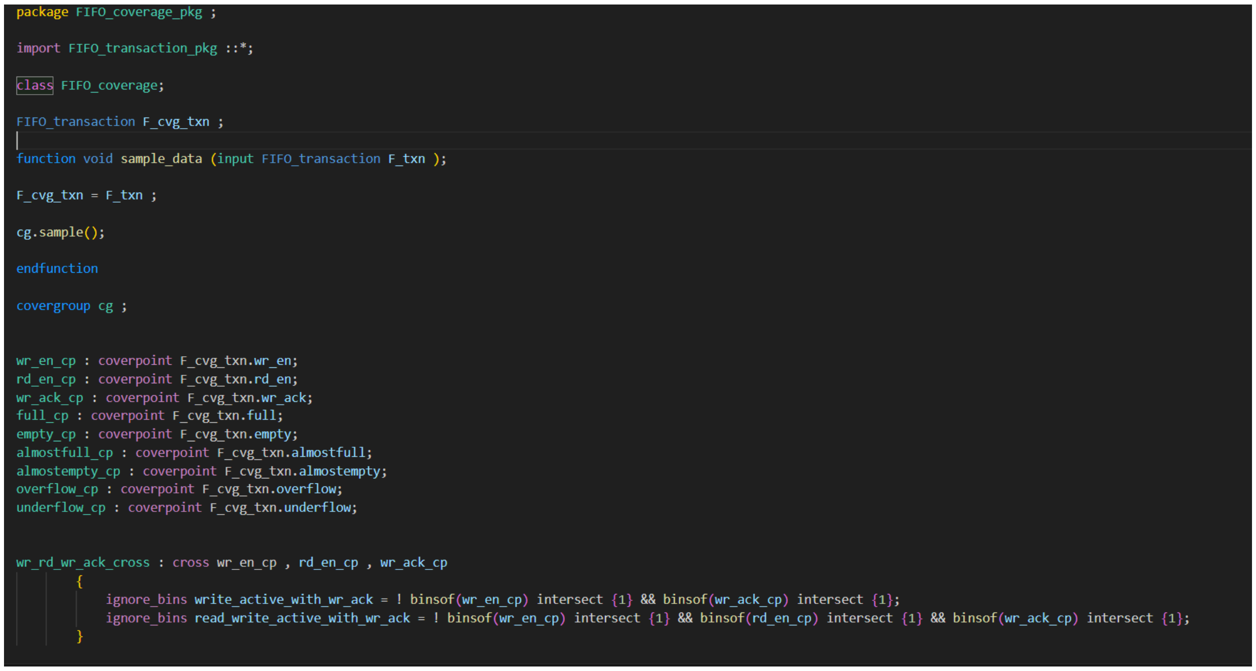
## Packages:

### Transaction package:

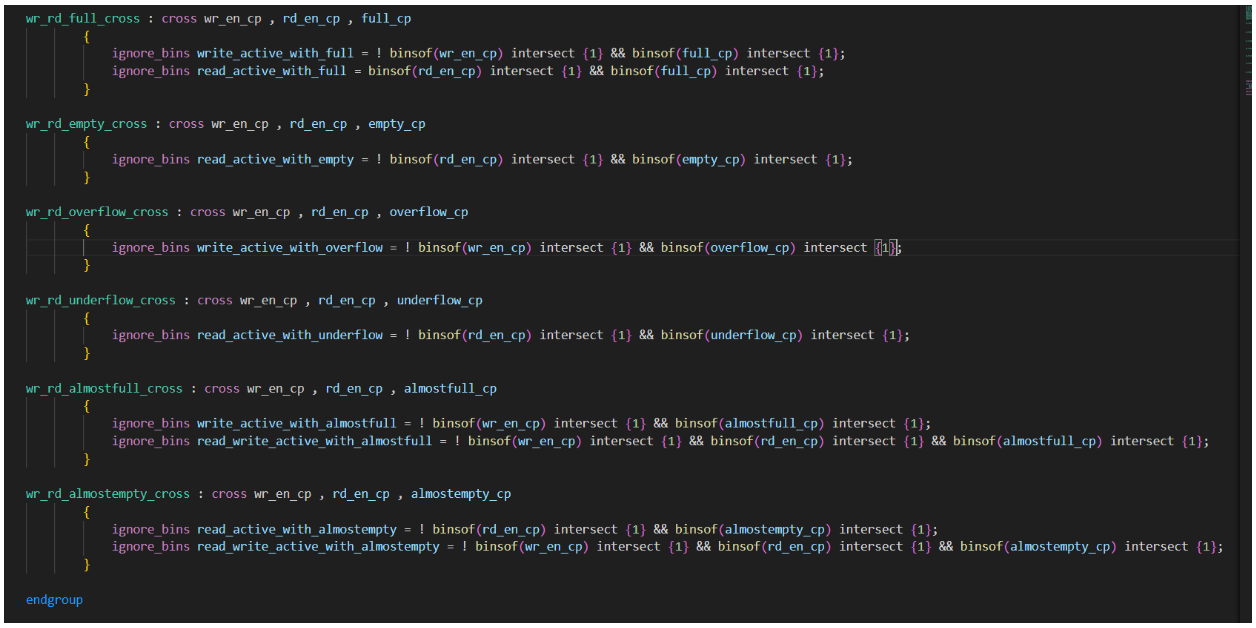


```
package FIFO_transaction_pkg ;

class FIFO_transaction;

parameter FIFO_WIDTH = 16 ;
parameter FIFO_DEPTH = 8 ;

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


int WR_EN_ON_DIST = 70 ;
int RD_EN_ON_DIST = 30 ;


constraint c_reset { rst_n dist{ 0:/2 , 1:/98 }; }

constraint c_write { wr_en dist { 0:/(100-WR_EN_ON_DIST)  , 1:/(WR_EN_ON_DIST) }; }

constraint c_read { rd_en dist { 0:/(100-RD_EN_ON_DIST) , 1:/(RD_EN_ON_DIST) }; }


constraint write_only { rst_n == 1;  wr_en == 1;  rd_en == 0; }


constraint read_only { rst_n == 1;  wr_en == 0;  rd_en == 1; }


endclass

endpackage
```

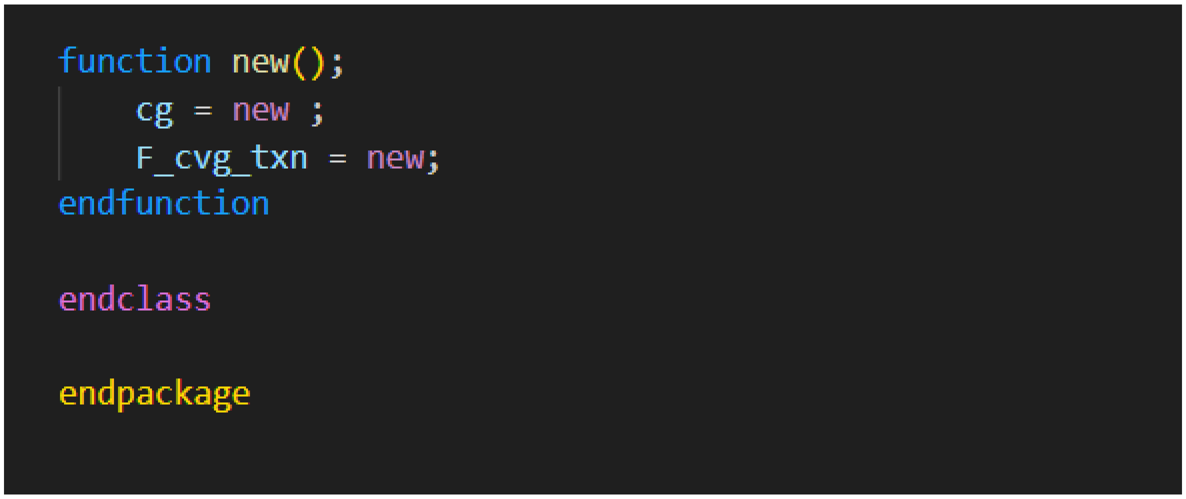
## Coverage package:



```
package FIFO_coverage_pkg ;

import FIFO_transaction_pkg ::*;

class FIFO_coverage;

FIFO_transaction F_cvg_txn ;

function void sample_data (input FIFO_transaction F_txn );

F_cvg_txn = F_txn ;

cg.sample();

endfunction

covergroup cg ;


wr_en_cp : coverpoint F_cvg_txn.wr_en;
rd_en_cp : coverpoint F_cvg_txn.rd_en;
wr_ack_cp : coverpoint F_cvg_txn.wr_ack;
full_cp : coverpoint F_cvg_txn.full;
empty_cp : coverpoint F_cvg_txn.empty;
almostfull_cp : coverpoint F_cvg_txn.almostfull;
almostempty_cp : coverpoint F_cvg_txn.almostempty;
overflow_cp : coverpoint F_cvg_txn.overflow;
underflow_cp : coverpoint F_cvg_txn.underflow;


wr_rd_wr_ack_cross : cross wr_en_cp , rd_en_cp , wr_ack_cp
        {
            ignore_bins write_active_with_wr_ack = ! binsof(wr_en_cp) intersect {1} && binsof(wr_ack_cp) intersect {1};
            ignore_bins read_write_active_with_wr_ack = ! binsof(wr_en_cp) intersect {1} && binsof(rd_en_cp) intersect {1} && binsof(wr_ack_cp) intersect {1};
        }
```

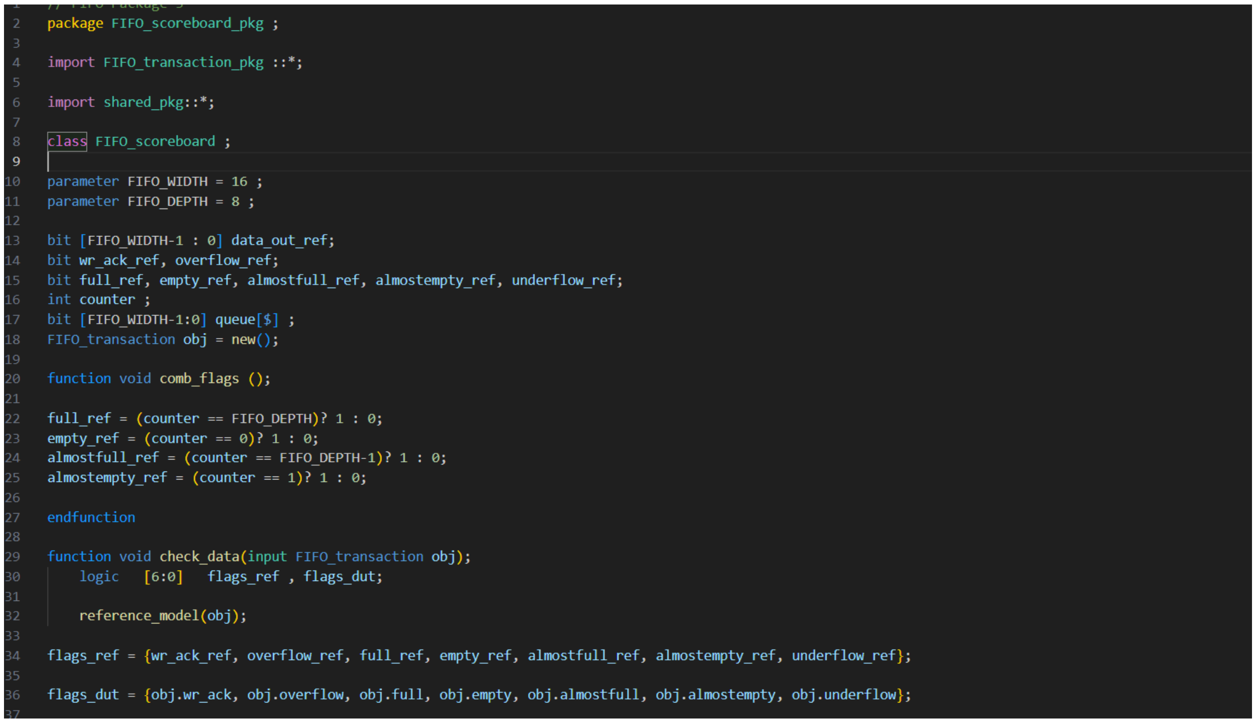


```
wr_rd_full_cross : cross wr_en_cp , rd_en_cp , full_cp
        {
            ignore_bins write_active_with_full = ! binsof(wr_en_cp) intersect {1} && binsof(full_cp) intersect {1};
            ignore_bins read_active_with_full = binsof(rd_en_cp) intersect {1} && binsof(full_cp) intersect {1};
        }

wr_rd_empty_cross : cross wr_en_cp , rd_en_cp , empty_cp
        {
            ignore_bins read_active_with_empty = ! binsof(rd_en_cp) intersect {1} && binsof(empty_cp) intersect {1};
        }

wr_rd_overflow_cross : cross wr_en_cp , rd_en_cp , overflow_cp
        {
            ignore_bins write_active_with_overflow = ! binsof(wr_en_cp) intersect {1} && binsof(overflow_cp) intersect {1};
        }

wr_rd_underflow_cross : cross wr_en_cp , rd_en_cp , underflow_cp
        {
            ignore_bins read_active_with_underflow = ! binsof(rd_en_cp) intersect {1} && binsof(underflow_cp) intersect {1};
        }

wr_rd_almostfull_cross : cross wr_en_cp , rd_en_cp , almostfull_cp
        {
            ignore_bins write_active_with_almostfull = ! binsof(wr_en_cp) intersect {1} && binsof(almostfull_cp) intersect {1};
            ignore_bins read_write_active_with_almostfull = ! binsof(wr_en_cp) intersect {1} && binsof(rd_en_cp) intersect {1} && binsof(almostfull_cp) intersect {1};
        }

wr_rd_almostempty_cross : cross wr_en_cp , rd_en_cp , almostempty_cp
        {
            ignore_bins read_active_with_almostempty = ! binsof(rd_en_cp) intersect {1} && binsof(almostempty_cp) intersect {1};
            ignore_bins read_write_active_with_almostempty = ! binsof(wr_en_cp) intersect {1} && binsof(rd_en_cp) intersect {1} && binsof(almostempty_cp) intersect {1};
        }

endgroup
```

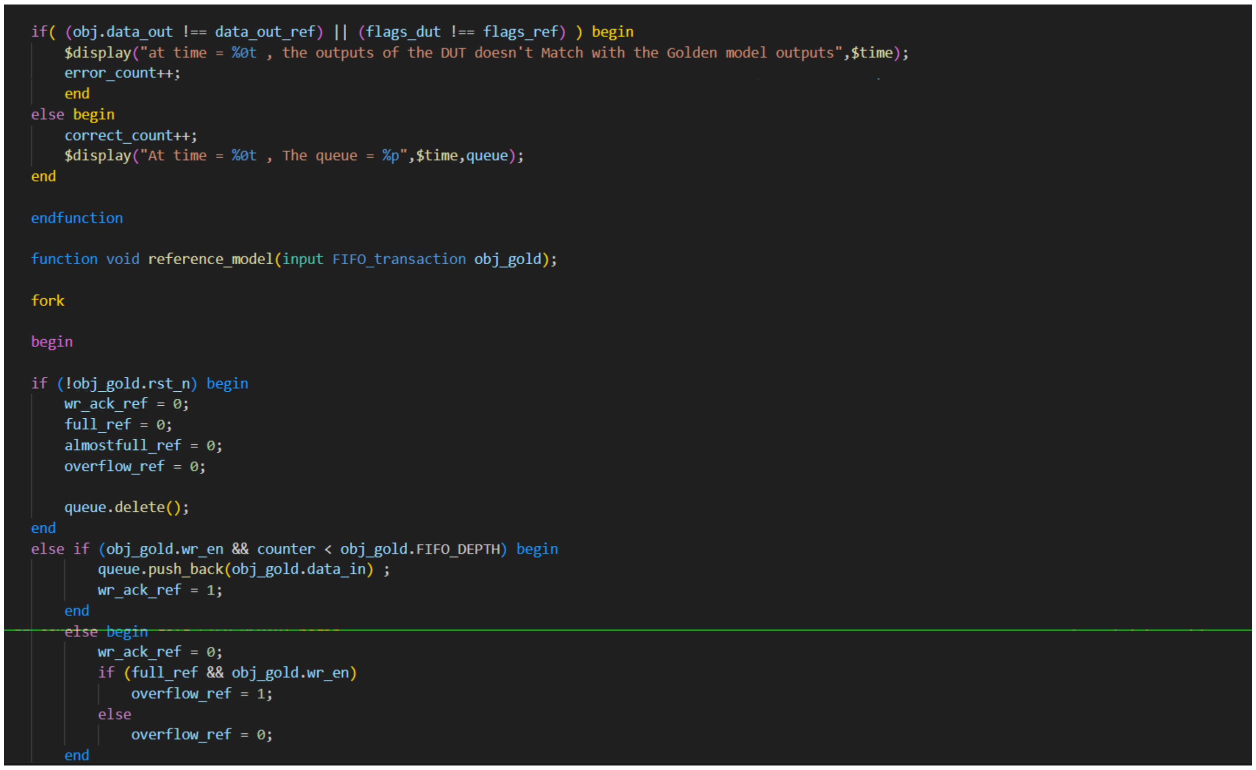


```
function new();
    cg = new ;
    F_cvg_txn = new;
endfunction

endclass

endpackage
```

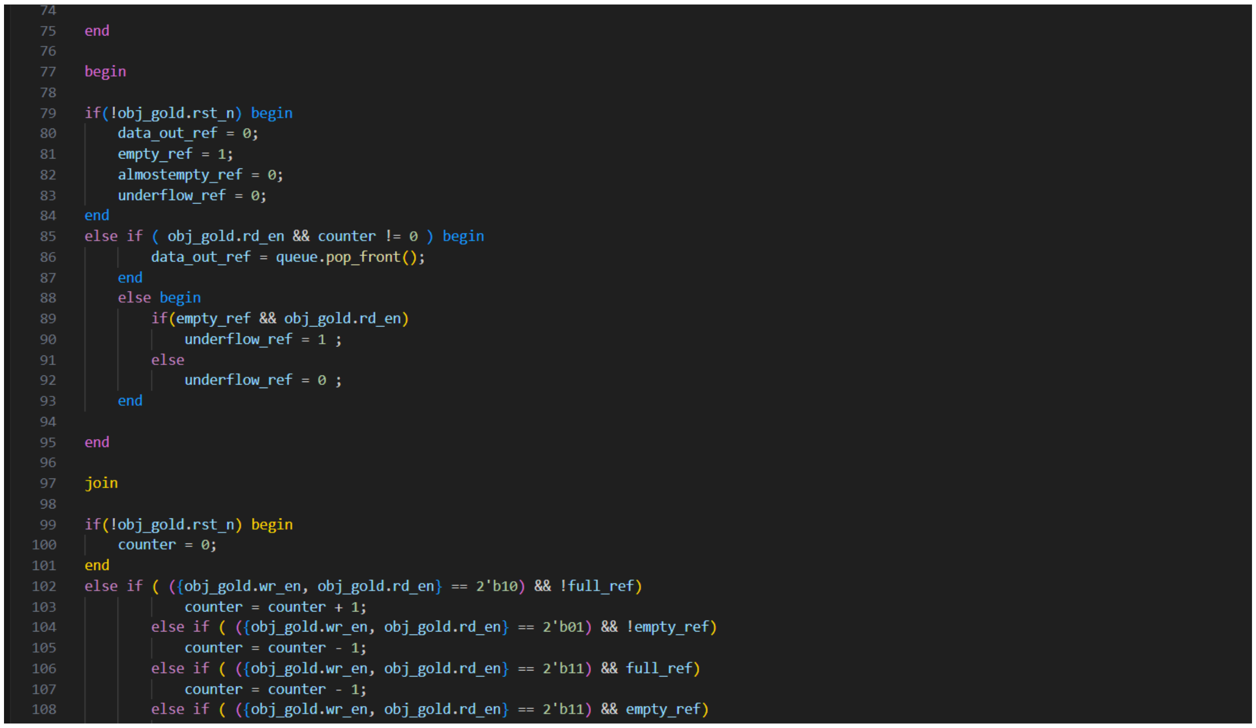
Scoreboard package:



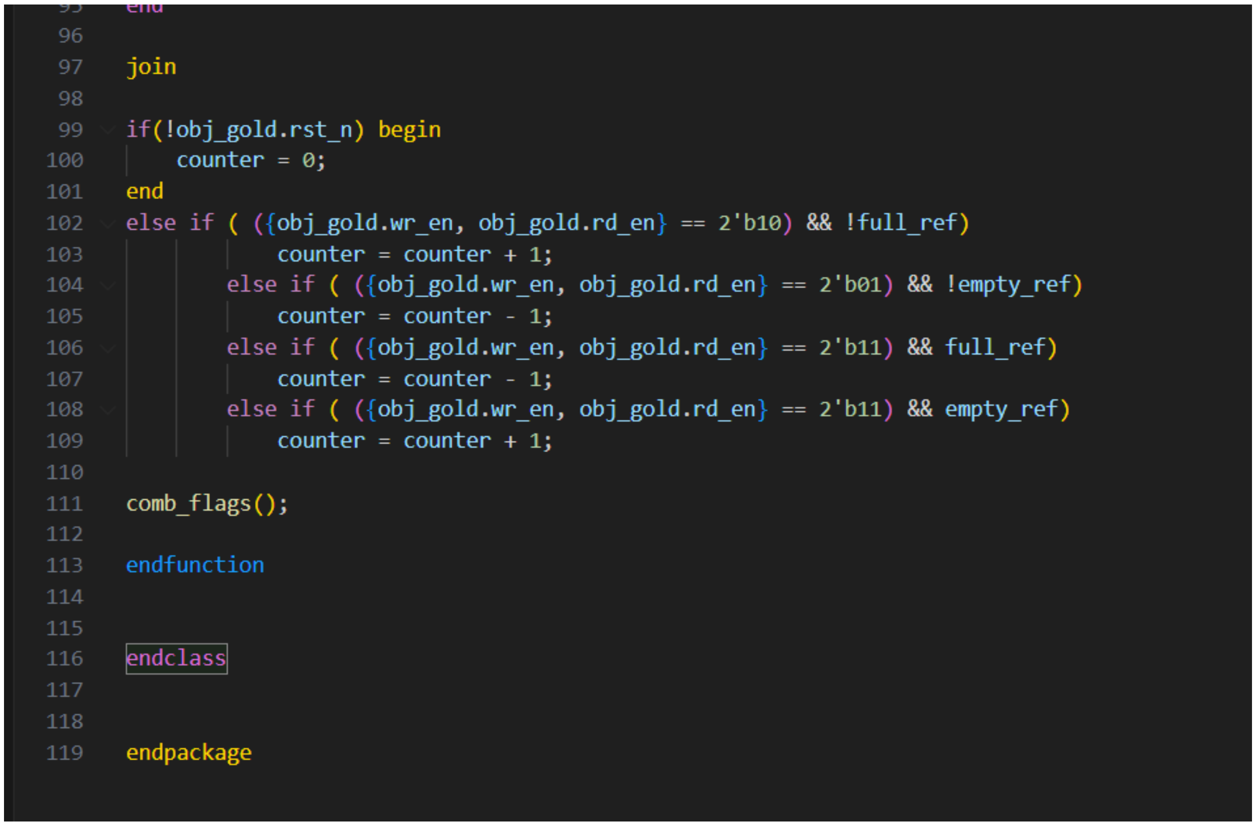
```
// FIFO Package 5
package FIFO_scoreboard_pkg ;

import FIFO_transaction_pkg ::*;

import shared_pkg::*;

class FIFO_scoreboard ;

parameter FIFO_WIDTH = 16 ;
parameter FIFO_DEPTH = 8 ;

bit [FIFO_WIDTH-1 : 0] data_out_ref;
bit wr_ack_ref, overflow_ref;
bit full_ref, empty_ref, almostfull_ref, almostempty_ref, underflow_ref;
int counter ;
bit [FIFO_WIDTH-1:0] queue[$] ;
FIFO_transaction obj = new();

function void comb_flags ();

full_ref = (counter == FIFO_DEPTH)? 1 : 0;
empty_ref = (counter == 0)? 1 : 0;
almostfull_ref = (counter == FIFO_DEPTH-1)? 1 : 0;
almostempty_ref = (counter == 1)? 1 : 0;

endfunction

function void check_data(input FIFO_transaction obj);
    logic   [6:0]   flags_ref , flags_dut;

    reference_model(obj);

flags_ref = {wr_ack_ref, overflow_ref, full_ref, empty_ref, almostfull_ref, almostempty_ref, underflow_ref};

flags_dut = {obj.wr_ack, obj.overflow, obj.full, obj.empty, obj.almostfull, obj.almostempty, obj.underflow};
```



```
if( (obj.data_out !== data_out_ref) || (flags_dut !== flags_ref) ) begin
    $display("at time = %0t , the outputs of the DUT doesn't Match with the Golden model outputs",$time);
    error_count++;
    end
else begin
    correct_count++;
    $display("At time = %0t , The queue = %p",$time,queue);
end

endfunction

function void reference_model(input FIFO_transaction obj_gold);

fork

begin

if (!obj_gold.rst_n) begin
    wr_ack_ref = 0;
    full_ref = 0;
    almostfull_ref = 0;
    overflow_ref = 0;

    queue.delete();
end
else if (obj_gold.wr_en && counter < obj_gold.FIFO_DEPTH) begin
        queue.push_back(obj_gold.data_in) ;
        wr_ack_ref = 1;
    end
    else begin
        wr_ack_ref = 0;
        if (full_ref && obj_gold.wr_en)
            overflow_ref = 1;
        else
            overflow_ref = 0;
    end
```

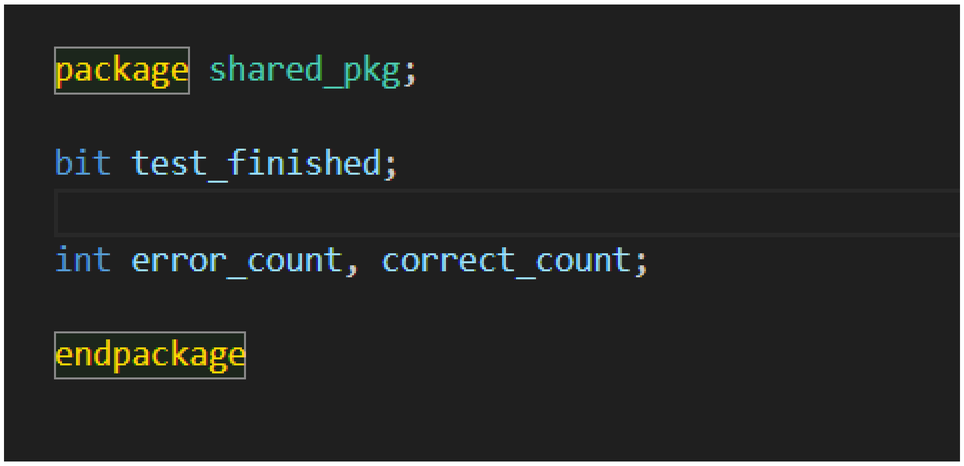


```
end

begin

if(!obj_gold.rst_n) begin
    data_out_ref = 0;
    empty_ref = 1;
    almostempty_ref = 0;
    underflow_ref = 0;
end
else if ( obj_gold.rd_en && counter != 0 ) begin
        data_out_ref = queue.pop_front();
    end
    else begin
        if(empty_ref && obj_gold.rd_en)
            underflow_ref = 1 ;
        else
            underflow_ref = 0 ;
    end

end

join

if(!obj_gold.rst_n) begin
    counter = 0;
end
else if ( ({obj_gold.wr_en, obj_gold.rd_en} == 2'b10) && !full_ref)
            counter = counter + 1;
        else if ( ({obj_gold.wr_en, obj_gold.rd_en} == 2'b01) && !empty_ref)
            counter = counter - 1;
        else if ( ({obj_gold.wr_en, obj_gold.rd_en} == 2'b11) && full_ref)
            counter = counter - 1;
        else if ( ({obj_gold.wr_en, obj_gold.rd_en} == 2'b11) && empty_ref)
```



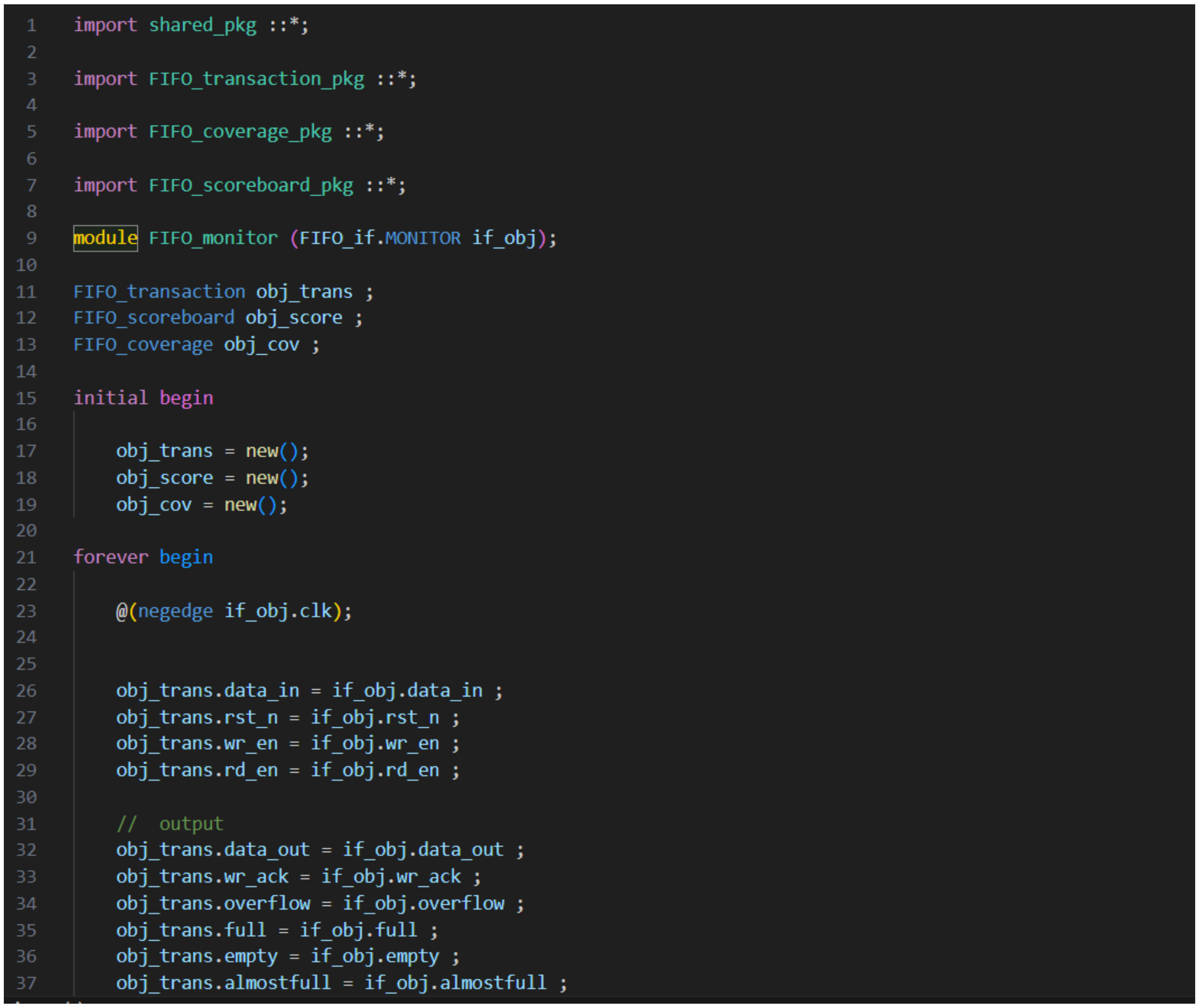
```
end

join

if(!obj_gold.rst_n) begin
    counter = 0;
end
else if ( ({obj_gold.wr_en, obj_gold.rd_en} == 2'b10) && !full_ref)
            counter = counter + 1;
        else if ( ({obj_gold.wr_en, obj_gold.rd_en} == 2'b01) && !empty_ref)
            counter = counter - 1;
        else if ( ({obj_gold.wr_en, obj_gold.rd_en} == 2'b11) && full_ref)
            counter = counter - 1;
        else if ( ({obj_gold.wr_en, obj_gold.rd_en} == 2'b11) && empty_ref)
            counter = counter + 1;

comb_flags();

endfunction


endclass


endpackage
```

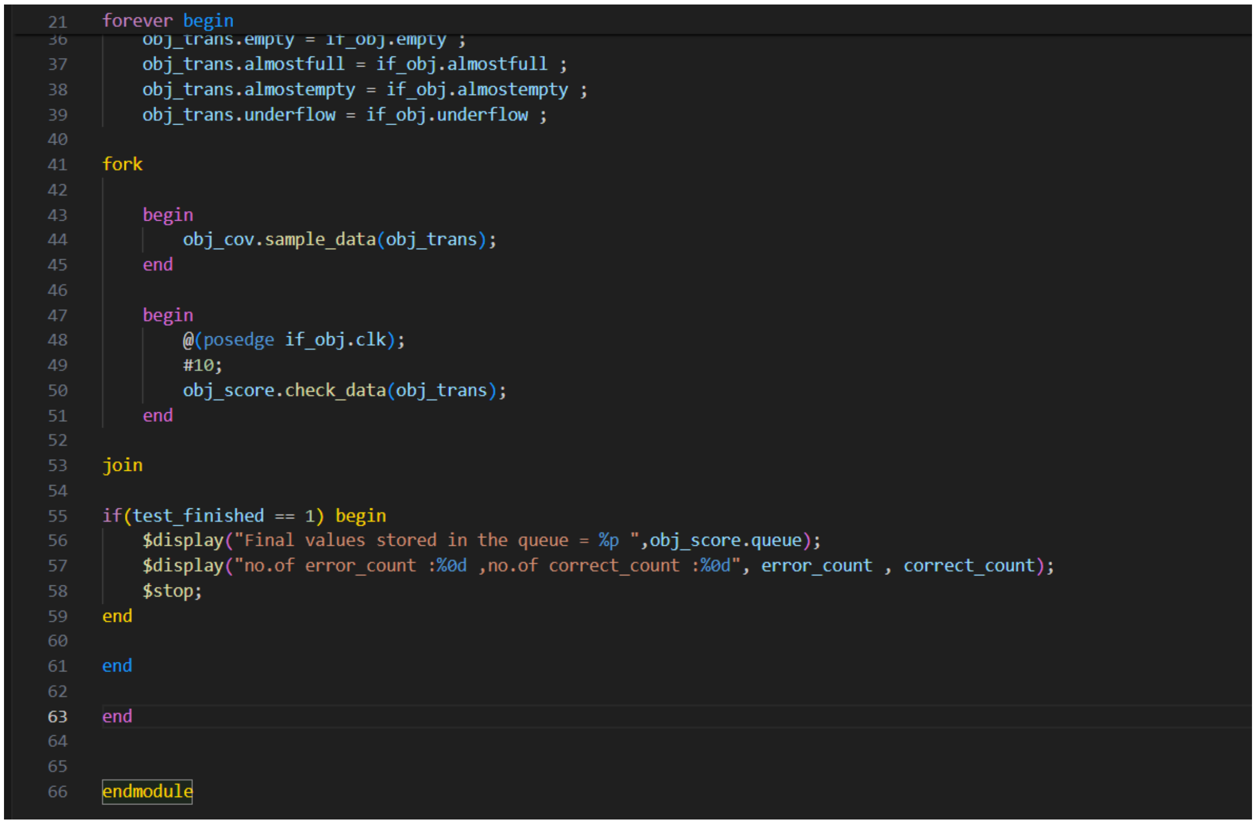
Shared package:



```
package shared_pkg;

bit test_finished;

int error_count, correct_count;

endpackage
```

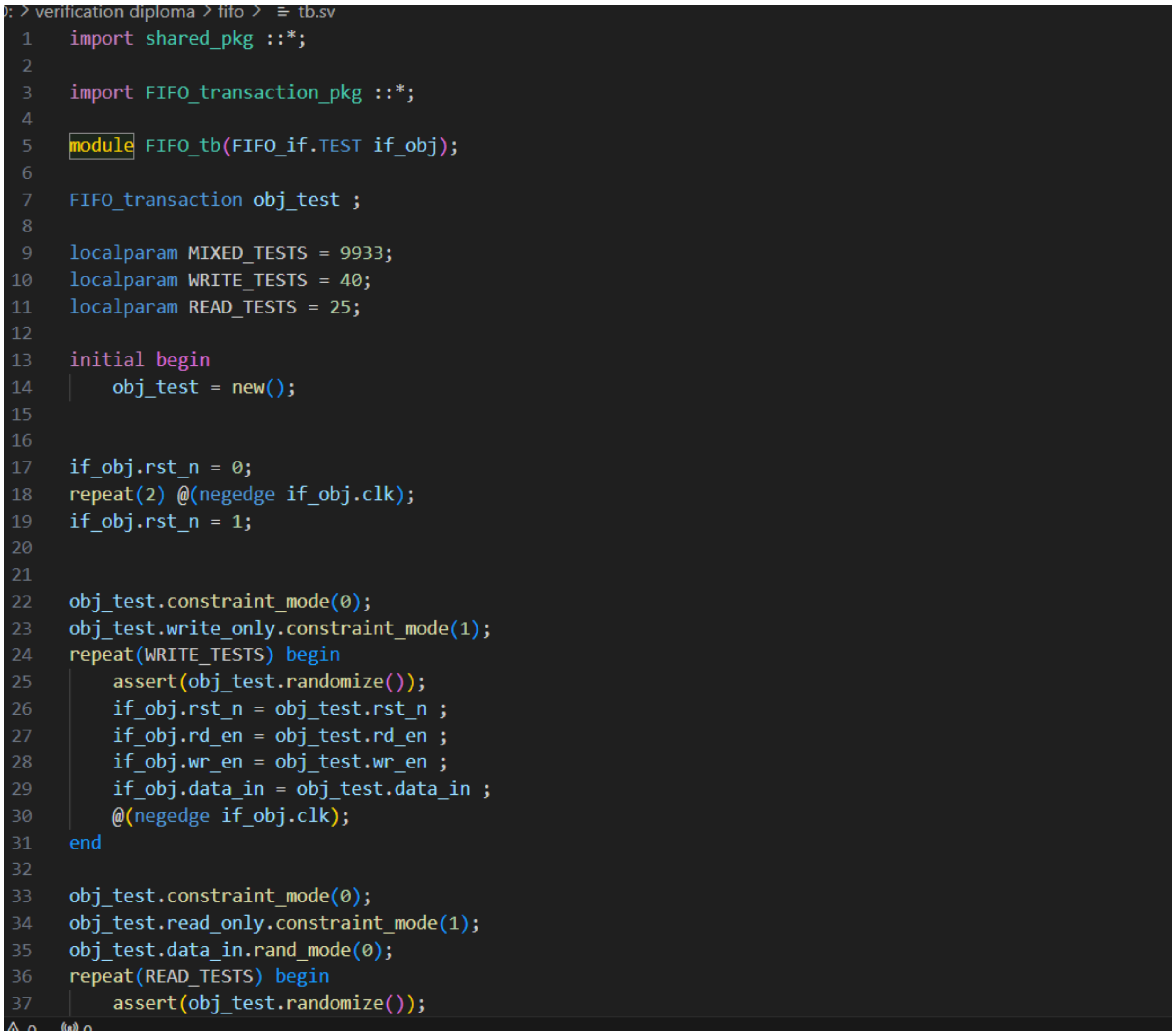
Monitor :


```
import shared_pkg ::*;

import FIFO_transaction_pkg ::*;

import FIFO_coverage_pkg ::*;

import FIFO_scoreboard_pkg ::*;

module FIFO_monitor (FIFO_if.MONITOR if_obj);

FIFO_transaction obj_trans ;
FIFO_scoreboard obj_score ;
FIFO_coverage obj_cov ;

initial begin

    obj_trans = new();
    obj_score = new();
    obj_cov = new();

forever begin

    @(negedge if_obj.clk);


    obj_trans.data_in = if_obj.data_in ;
    obj_trans.rst_n = if_obj.rst_n ;
    obj_trans.wr_en = if_obj.wr_en ;
    obj_trans.rd_en = if_obj.rd_en ;

    //  output
    obj_trans.data_out = if_obj.data_out ;
    obj_trans.wr_ack = if_obj.wr_ack ;
    obj_trans.overflow = if_obj.overflow ;
    obj_trans.full = if_obj.full ;
    obj_trans.empty = if_obj.empty ;
    obj_trans.almostfull = if_obj.almostfull ;
```



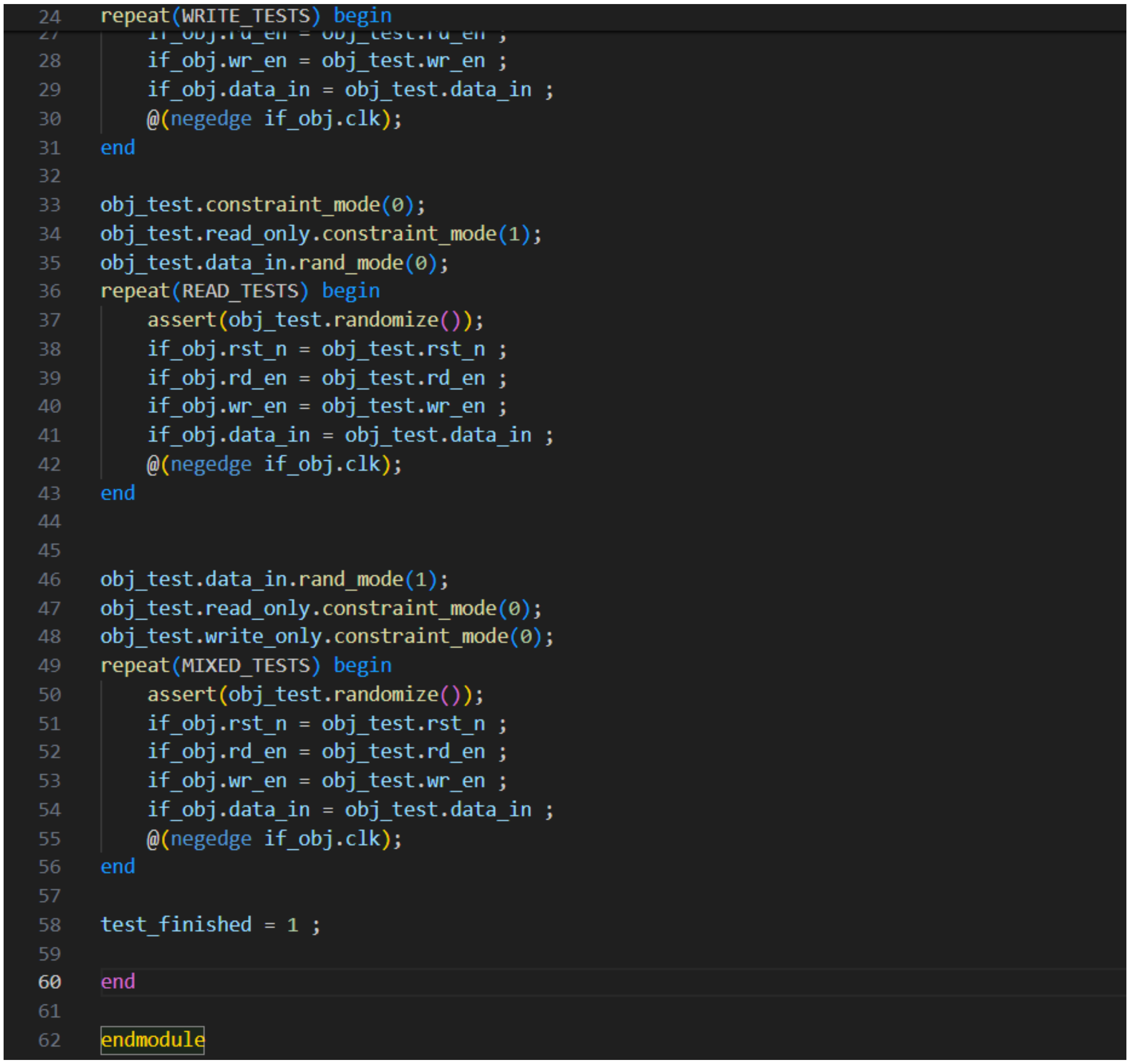
```
forever begin
    obj_trans.empty = if_obj.empty ;
    obj_trans.almostfull = if_obj.almostfull ;
    obj_trans.almostempty = if_obj.almostempty ;
    obj_trans.underflow = if_obj.underflow ;

fork

    begin
        obj_cov.sample_data(obj_trans);
    end

    begin
        @(posedge if_obj.clk);
        #10;
        obj_score.check_data(obj_trans);
    end

join

if(test_finished == 1) begin
    $display("Final values stored in the queue = %p ",obj_score.queue);
    $display("no.of error_count :%0d ,no.of correct_count :%0d", error_count , correct_count);
    $stop;
end

end

end


endmodule
```

TB:



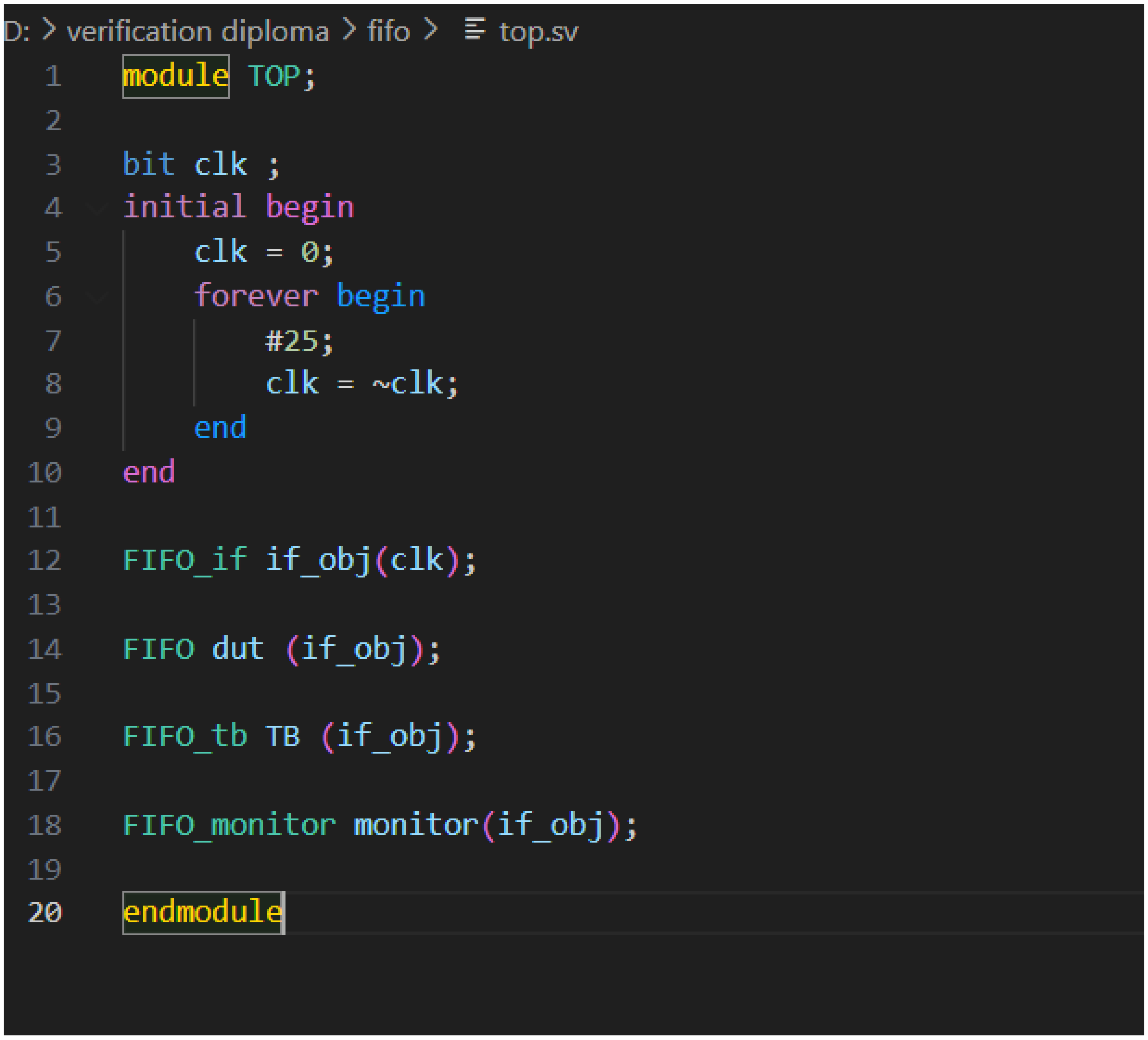
verification diploma > fifo > tb.sv

```
import shared_pkg ::*;

import FIFO_transaction_pkg ::*;

module FIFO_tb(FIFO_if.TEST if_obj);

FIFO_transaction obj_test ;

localparam MIXED_TESTS = 9933;
localparam WRITE_TESTS = 40;
localparam READ_TESTS = 25;

initial begin
    obj_test = new();


if_obj.rst_n = 0;
repeat(2) @(negedge if_obj.clk);
if_obj.rst_n = 1;


obj_test.constraint_mode(0);
obj_test.write_only.constraint_mode(1);
repeat(WRITE_TESTS) begin
    assert(obj_test.randomize());
    if_obj.rst_n = obj_test.rst_n ;
    if_obj.rd_en = obj_test.rd_en ;
    if_obj.wr_en = obj_test.wr_en ;
    if_obj.data_in = obj_test.data_in ;
    @(negedge if_obj.clk);
end

obj_test.constraint_mode(0);
obj_test.read_only.constraint_mode(1);
obj_test.data_in.rand_mode(0);
repeat(READ_TESTS) begin
    assert(obj_test.randomize());
```



```
repeat(WRITE_TESTS) begin
    if_obj.rd_en = obj_test.rd_en ;
    if_obj.wr_en = obj_test.wr_en ;
    if_obj.data_in = obj_test.data_in ;
    @(negedge if_obj.clk);
end

obj_test.constraint_mode(0);
obj_test.read_only.constraint_mode(1);
obj_test.data_in.rand_mode(0);
repeat(READ_TESTS) begin
    assert(obj_test.randomize());
    if_obj.rst_n = obj_test.rst_n ;
    if_obj.rd_en = obj_test.rd_en ;
    if_obj.wr_en = obj_test.wr_en ;
    if_obj.data_in = obj_test.data_in ;
    @(negedge if_obj.clk);
end


obj_test.data_in.rand_mode(1);
obj_test.read_only.constraint_mode(0);
obj_test.write_only.constraint_mode(0);
repeat(MIXED_TESTS) begin
    assert(obj_test.randomize());
    if_obj.rst_n = obj_test.rst_n ;
    if_obj.rd_en = obj_test.rd_en ;
    if_obj.wr_en = obj_test.wr_en ;
    if_obj.data_in = obj_test.data_in ;
    @(negedge if_obj.clk);
end

test_finished = 1 ;

end

endmodule
```

TOP:



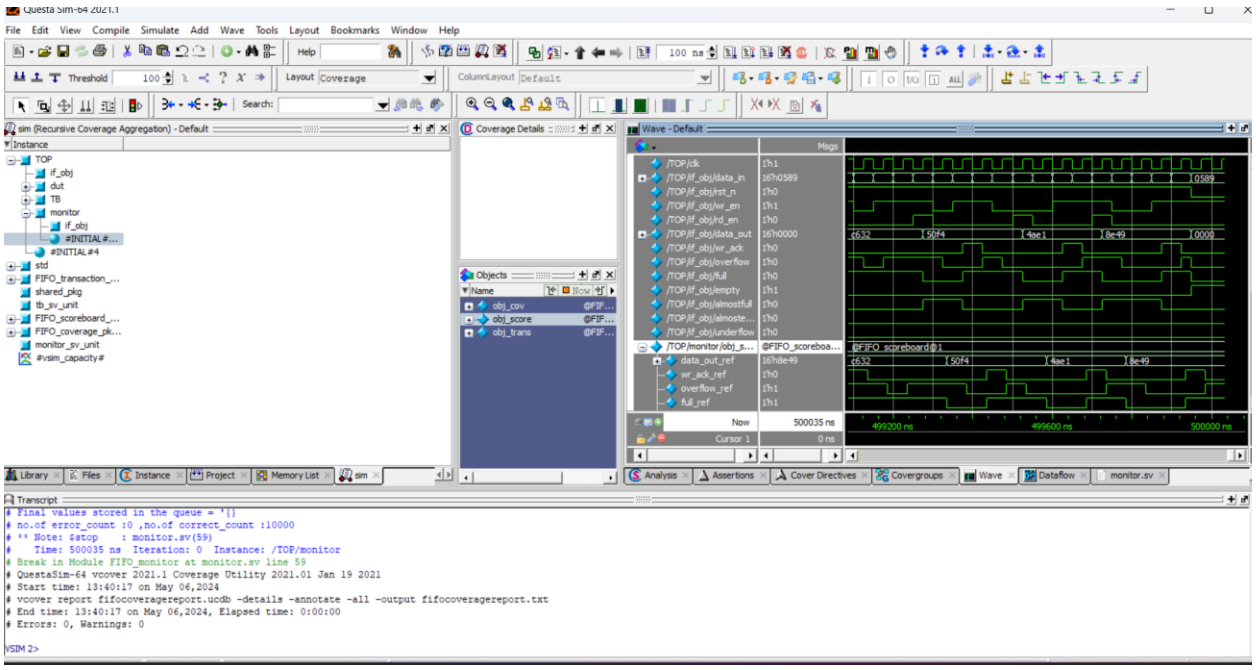
D: > verification diploma > fifo > top.sv

```
module TOP;

bit clk ;
initial begin
    clk = 0;
    forever begin
        #25;
        clk = ~clk;
    end
end

FIFO_if if_obj(clk);

FIFO dut (if_obj);

FIFO_tb TB (if_obj);

FIFO_monitor monitor(if_obj);

endmodule
```

## DO file:

```
vlib work
vlog FIFO.sv package_1.sv package_2.sv package_3.sv shared_pkg.sv tb.sv monitor.sv interface.sv top.sv +cover
vsim -voptargs=+acc work.TOP -cover
add wave *
add wave -position insertpoint  \
sim:/TOP/if_obj/data_in \
sim:/TOP/if_obj/rst_n \
sim:/TOP/if_obj/wr_en \
sim:/TOP/if_obj/rd_en \
sim:/TOP/if_obj/data_out \
sim:/TOP/if_obj/wr_ack \
sim:/TOP/if_obj/overflow \
sim:/TOP/if_obj/full \
sim:/TOP/if_obj/empty \
sim:/TOP/if_obj/almostfull \
sim:/TOP/if_obj/almostempty \
sim:/TOP/if_obj/underflow \
sim:/TOP/monitor/obj_score
add wave /TOP/dut/write_acknowledge /TOP/dut/underflow_assertion /TOP/dut/overflow_assertion /TOP/dut/increment_assertion
/TOP/dut/decrement_assertion /TOP/dut/full_assertion /TOP/dut/empty_assertion /TOP/dut/almostfull_assertion /TOP/dut/almostempty_assertion
coverage save fifocoveragereport.ucdb -onexit -du work.TOP
run -all
coverage report -detail -cvg -directive -comments -output fcover_report.txt {}
#quit -sim
vcover report fifocoveragereport.ucdb -details -annotate -all -output fifocoveragereport.txt
```

## Questa snippet:



```
# Final values stored in the queue = '{}
# no.of error_count :0 ,no.of correct_count :10000
# ** Note: $stop    : monitor.sv(59)
#    Time: 500035 ns  Iteration: 0  Instance: /TOP/monitor
# Break in Module FIFO_monitor at monitor.sv line 59
# QuestaSim-64 vcover 2021.1 Coverage Utility 2021.01 Jan 19 2021
# Start time: 13:40:17 on May 06,2024
# vcover report fifocoveragereport.ucdb -details -annotate -all -output fifocoveragereport.txt
# End time: 13:40:17 on May 06,2024, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0

VSIM 2>
```