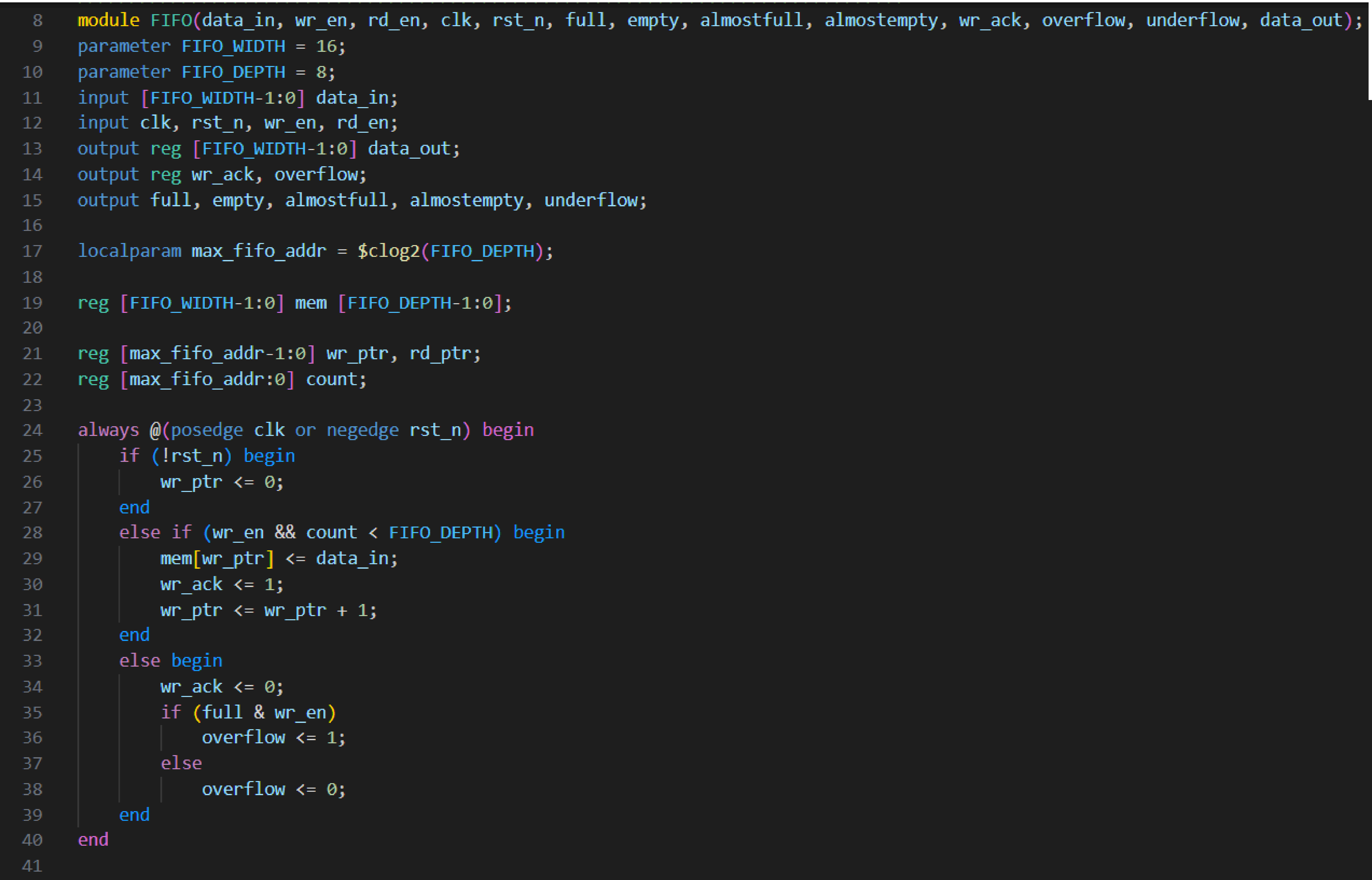
FIFO_SV verification

Presented by

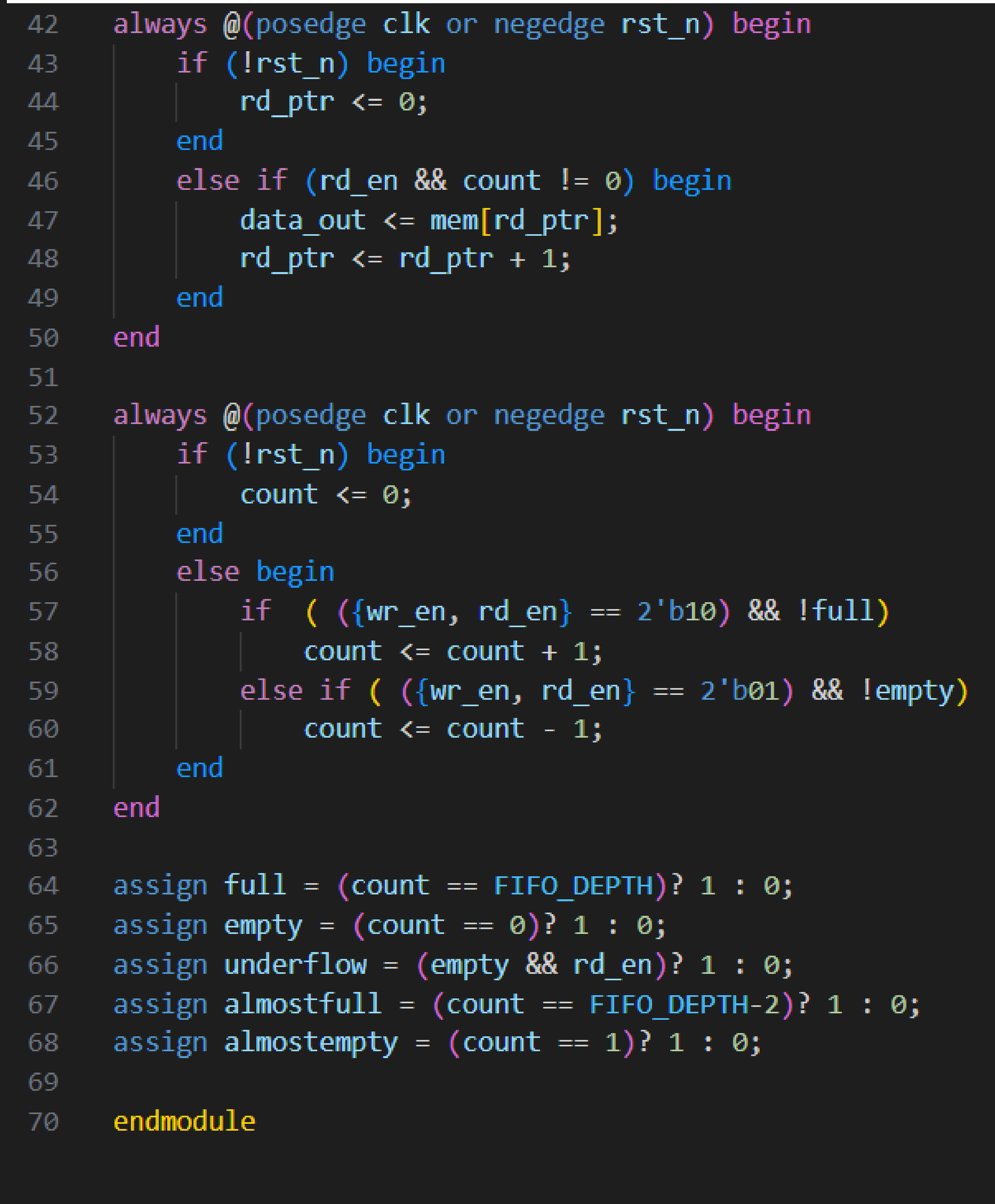
Ahmed Alaa
Abdelrahman

GITHUB

Original FIFO Design



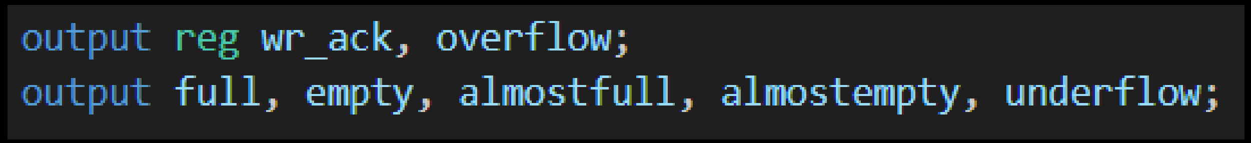
```verilog
module FIFO(data_in, wr_en, rd_en, clk, rst_n, full, empty, almostfull, almostempty, wr_ack, overflow, underflow, data_out);
parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;
input [FIFO_WIDTH-1:0] data_in;
input clk, rst_n, wr_en, rd_en;
output reg [FIFO_WIDTH-1:0] data_out;
output reg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

```



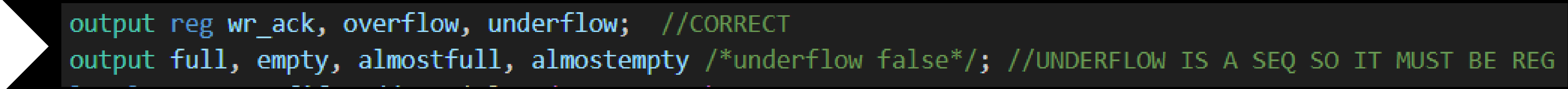
```verilog
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        rd_ptr <= 0;
    end
    else if (rd_en && count != 0) begin
        data_out <= mem[rd_ptr];
        rd_ptr <= rd_ptr + 1;
    end
end

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
```

Bugs and Fixes

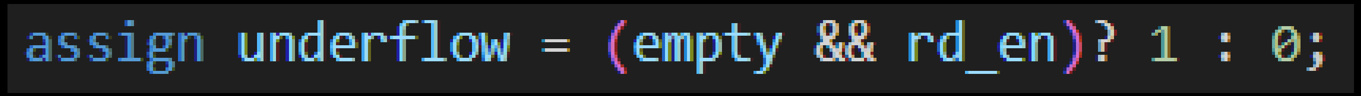
1-Under_Flow is a seq output



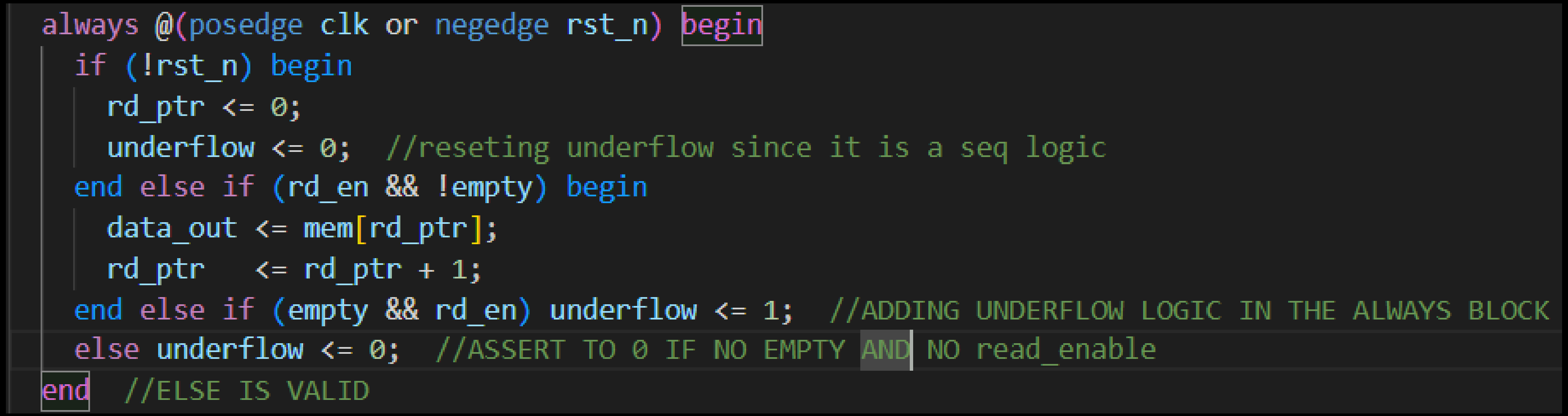
```
output reg wr_ack, overflow;
output full, empty, almostfull, almostempty, underflow;
```



```
output reg wr_ack, overflow, underflow;  //CORRECT
output full, empty, almostfull, almostempty /*underflow false*/; //UNDERFLOW IS A SEQ SO IT MUST BE REG
```

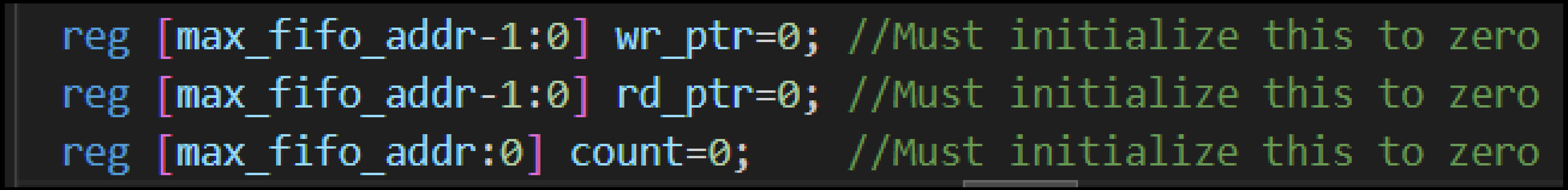


```
assign underflow = (empty && rd_en)? 1 : 0;
```



```
always @(posedge clk or negedge rst_n) begin
  if (!rst_n) begin
    rd_ptr <= 0;
    underflow <= 0;  //reseting underflow since it is a seq logic
  end else if (rd_en && !empty) begin
    data_out <= mem[rd_ptr];
    rd_ptr   <= rd_ptr + 1;
  end else if (empty && rd_en) underflow <= 1;  //ADDING UNDERFLOW LOGIC IN THE ALWAYS BLOCK
  else underflow <= 0;  //ASSERT TO 0 IF NO EMPTY AND NO read_enable
end  //ELSE IS VALID
```

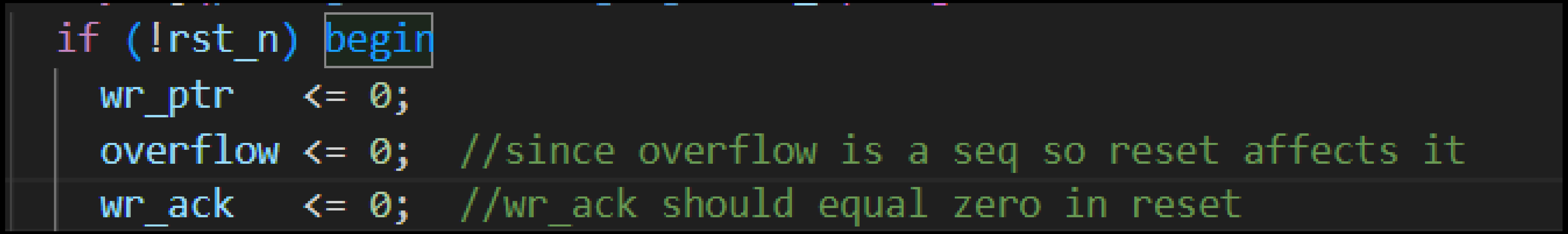
2-initialization Issue



```
  reg [max_fifo_addr-1:0] wr_ptr=0; //Must initialize this to zero
  reg [max_fifo_addr-1:0] rd_ptr=0; //Must initialize this to zero
  reg [max_fifo_addr:0] count=0;    //Must initialize this to zero
```

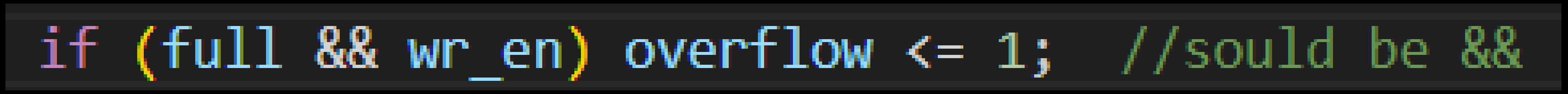
Bugs and Fixes

3-overflow and wr_ack must be reseted to zero



```
if (!rst_n) begin
  wr_ptr   <= 0;
  overflow <= 0;  //since overflow is a seq so reset affects it
  wr_ack   <= 0;  //wr_ack should equal zero in reset
```

4-To make the design hit the 100% conditional coverage we must use && instead of &

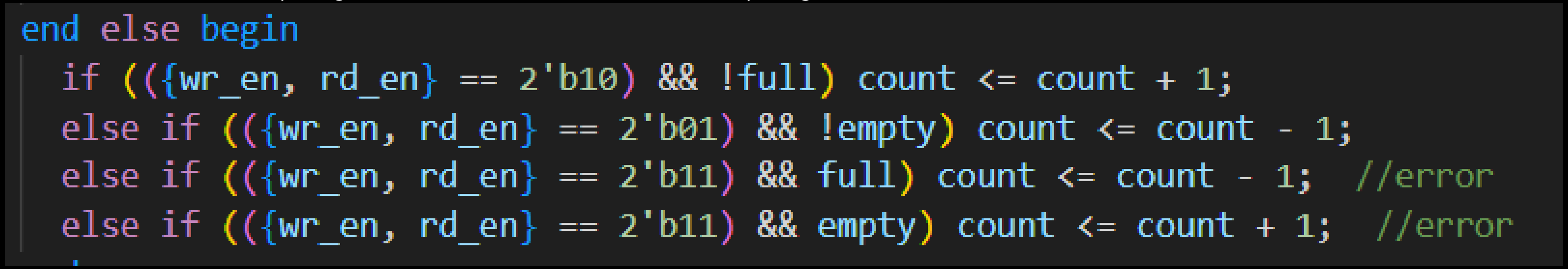


```
if (full && wr_en) overflow <= 1;  //sould be &&
```

Bugs and Fixes

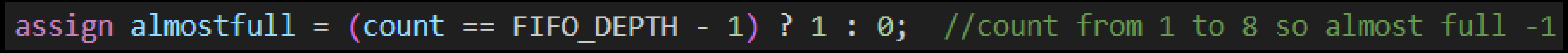
5-We must add more cases

when wr_en and rd_en are asserted count must change depending is it full or empty and if !full ||!empty then count remains constant



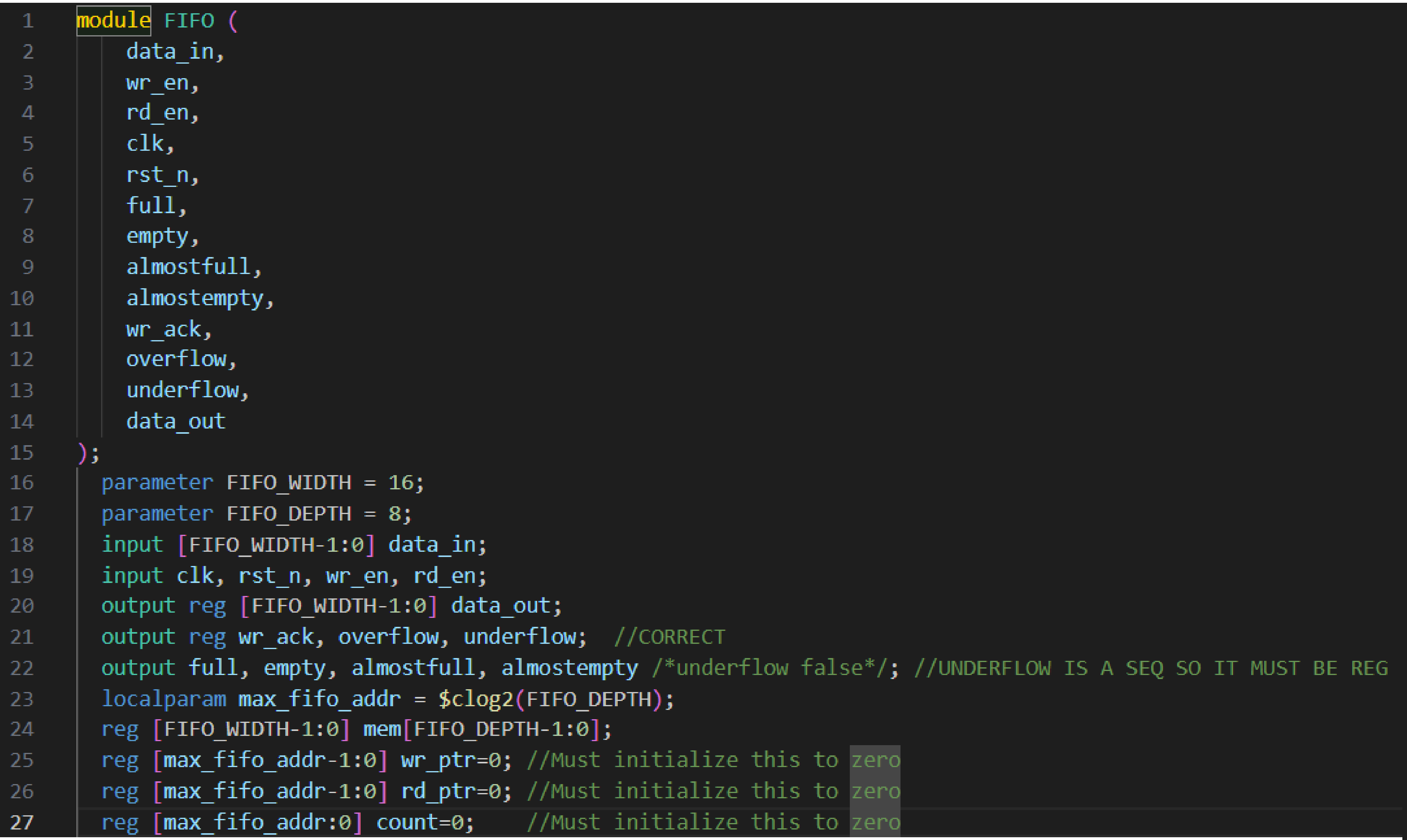
```
end else begin
  if (({wr_en, rd_en} == 2'b10) && !full) count <= count + 1;
  else if (({wr_en, rd_en} == 2'b01) && !empty) count <= count - 1;
  else if (({wr_en, rd_en} == 2'b11) && full) count <= count - 1;  //error
  else if (({wr_en, rd_en} == 2'b11) && empty) count <= count + 1;  //error
```

6-Almostfull is when the count = FIFO_DEPTH -1

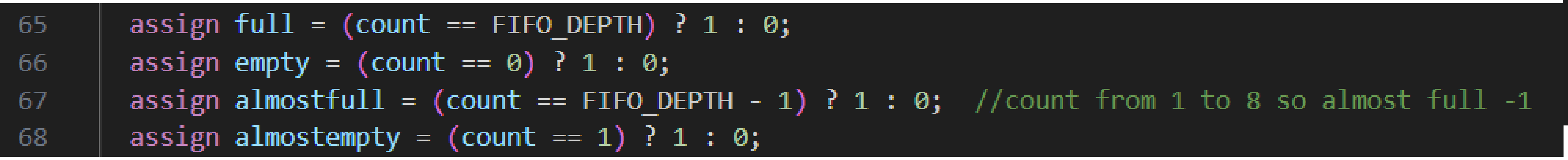


```
assign almostfull = (count == FIFO_DEPTH - 1) ? 1 : 0;  //count from 1 to 8 so almost full -1
```

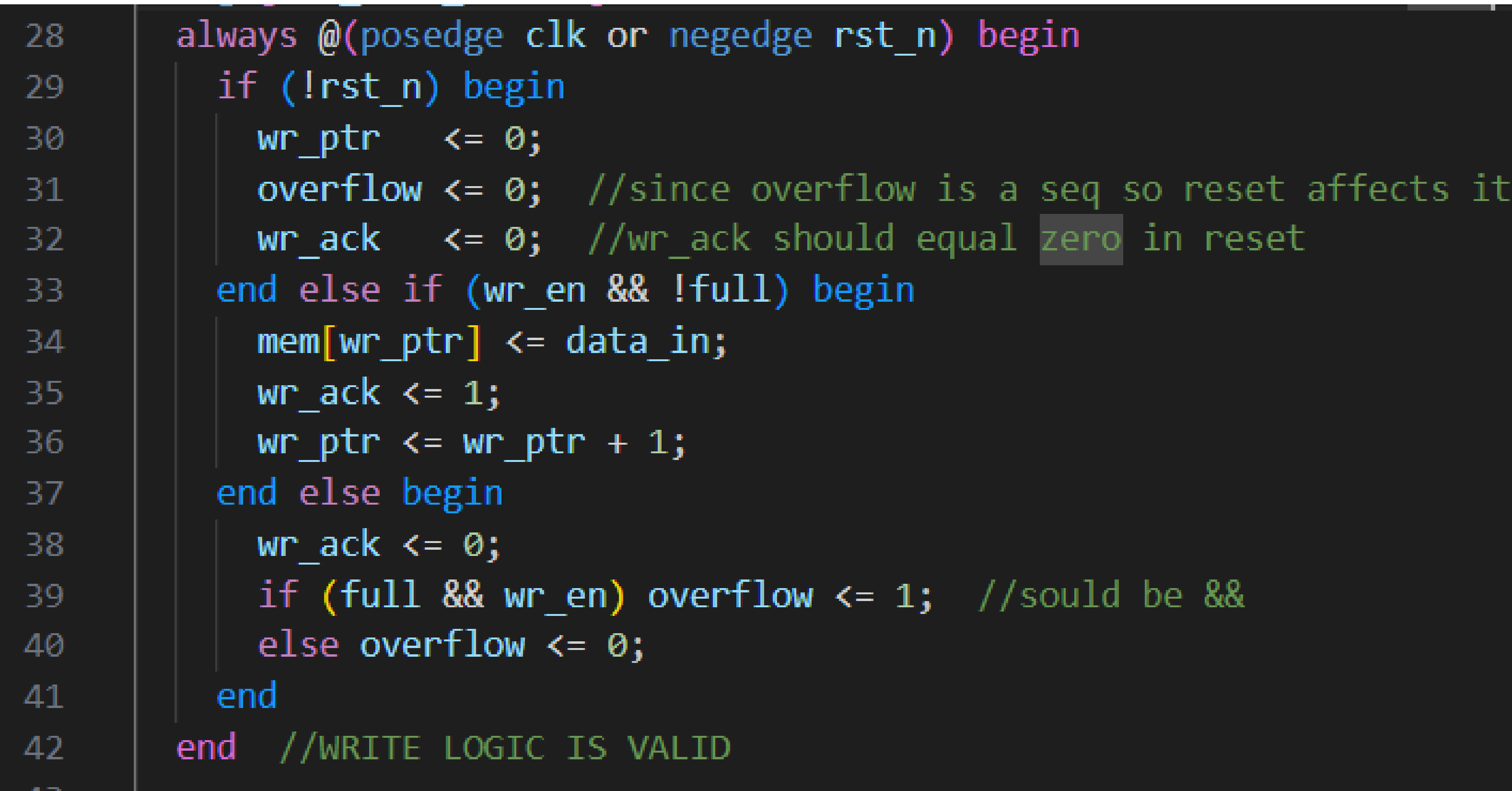
FIFO DESIGN AFTER



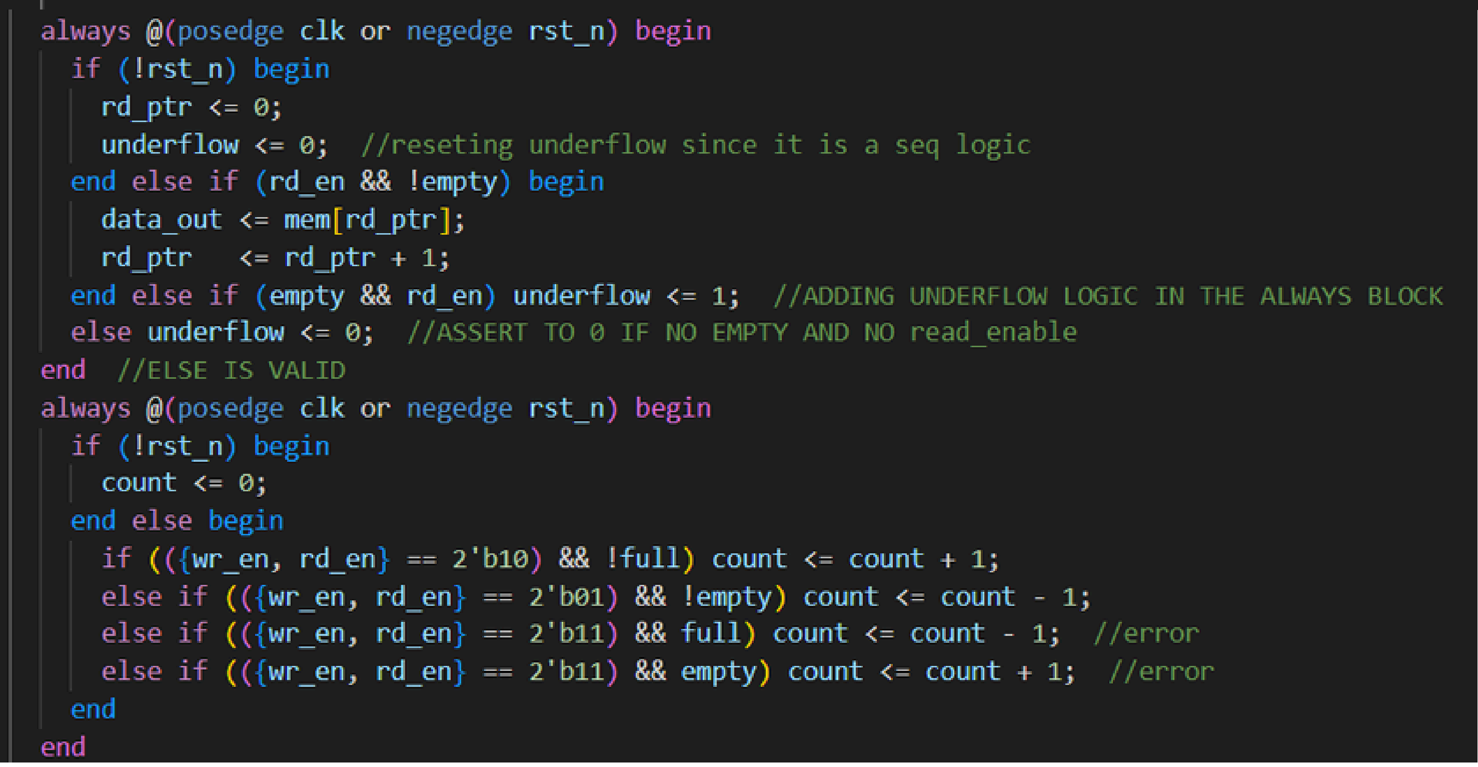
```
module FIFO (
    data_in,
    wr_en,
    rd_en,
    clk,
    rst_n,
    full,
    empty,
    almostfull,
    almostempty,
    wr_ack,
    overflow,
    underflow,
    data_out
);
  parameter FIFO_WIDTH = 16;
  parameter FIFO_DEPTH = 8;
  input [FIFO_WIDTH-1:0] data_in;
  input clk, rst_n, wr_en, rd_en;
  output reg [FIFO_WIDTH-1:0] data_out;
  output reg wr_ack, overflow, underflow;  //CORRECT
  output full, empty, almostfull, almostempty /*underflow false*/; //UNDERFLOW IS A SEQ SO IT MUST BE REG
  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=0; //Must initialize this to zero
  reg [max_fifo_addr-1:0] rd_ptr=0; //Must initialize this to zero
  reg [max_fifo_addr:0] count=0;    //Must initialize this to zero
```



```
  assign full = (count == FIFO_DEPTH) ? 1 : 0;
  assign empty = (count == 0) ? 1 : 0;
  assign almostfull = (count == FIFO_DEPTH - 1) ? 1 : 0;  //count from 1 to 8 so almost full -1
  assign almostempty = (count == 1) ? 1 : 0;
```

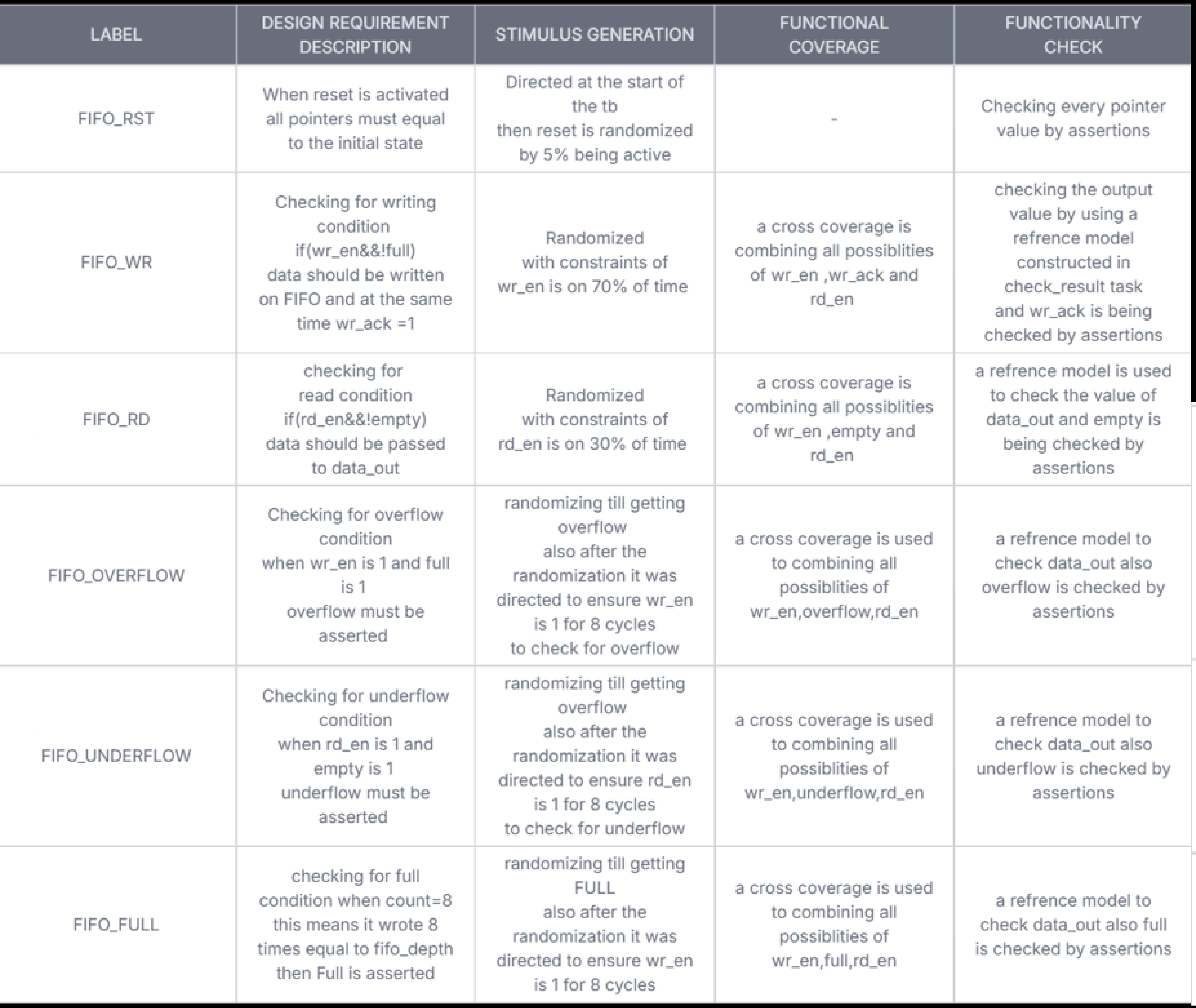


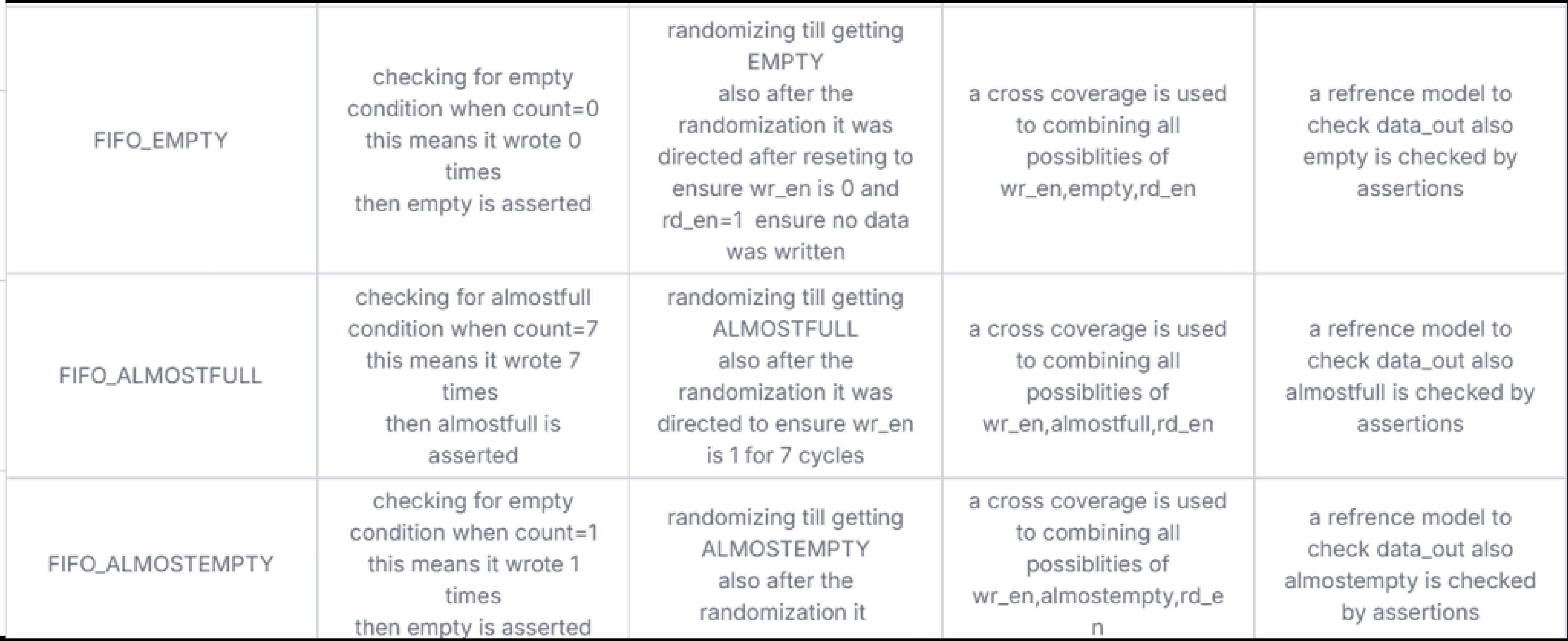
```
  always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
      wr_ptr   <= 0;
      overflow <= 0;  //since overflow is a seq so reset affects it
      wr_ack   <= 0;  //wr_ack should equal zero in reset
    end else if (wr_en && !full) begin
      mem[wr_ptr] <= data_in;
      wr_ack <= 1;
      wr_ptr <= wr_ptr + 1;
    end else begin
      wr_ack <= 0;
      if (full && wr_en) overflow <= 1;  //sould be &&
      else overflow <= 0;
    end
  end  //WRITE LOGIC IS VALID
```



```
  always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
      rd_ptr <= 0;
      underflow <= 0;  //reseting underflow since it is a seq logic
    end else if (rd_en && !empty) begin
      data_out <= mem[rd_ptr];
      rd_ptr   <= rd_ptr + 1;
    end else if (empty && rd_en) underflow <= 1;  //ADDING UNDERFLOW LOGIC IN THE ALWAYS BLOCK
    else underflow <= 0;  //ASSERT TO 0 IF NO EMPTY AND NO read_enable
  end  //ELSE IS VALID
  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;
      else if (({wr_en, rd_en} == 2'b11) && full) count <= count - 1;  //error
      else if (({wr_en, rd_en} == 2'b11) && empty) count <= count + 1;  //error
    end
  end
```

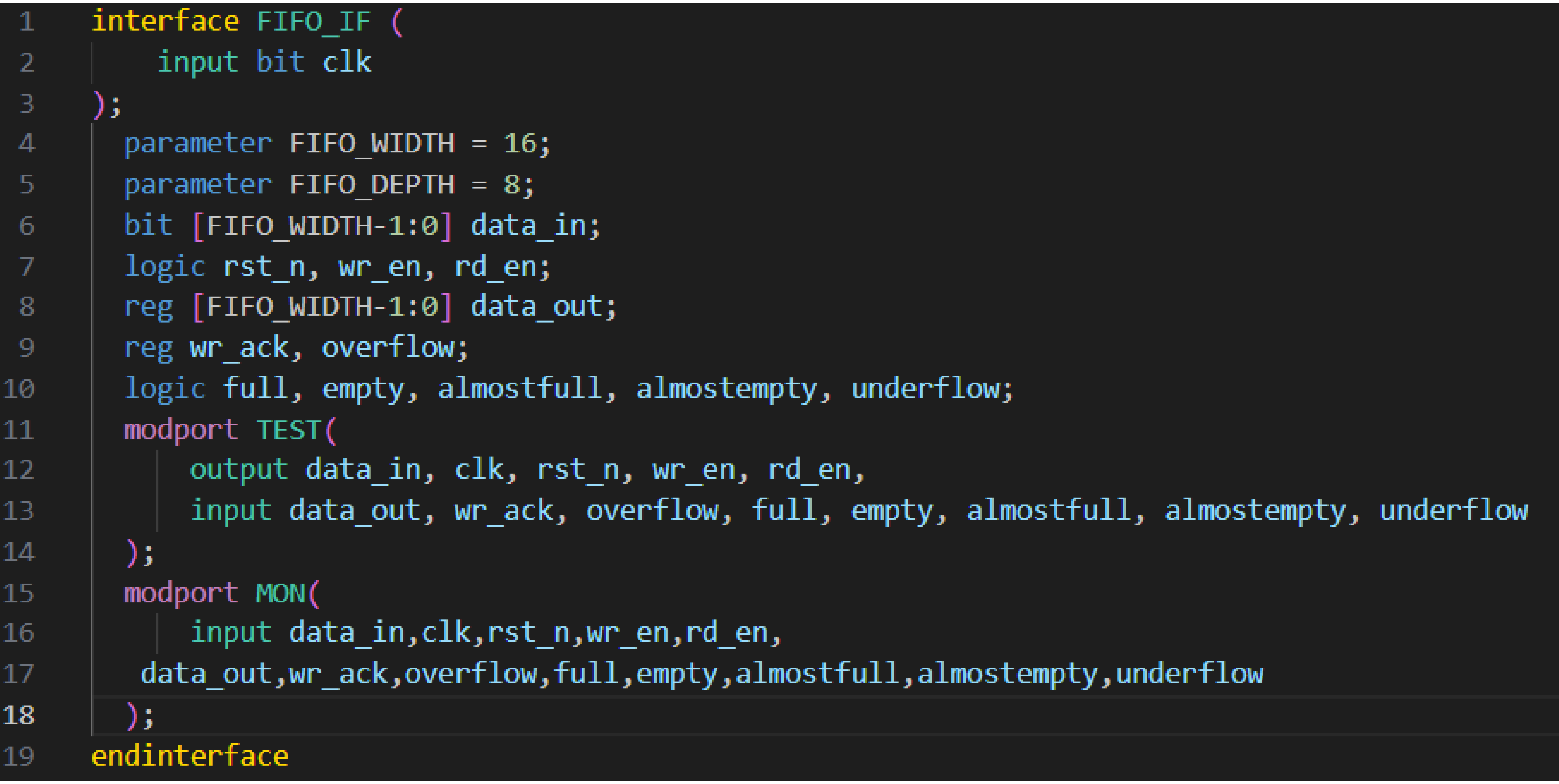
VERIFICATION PLAN



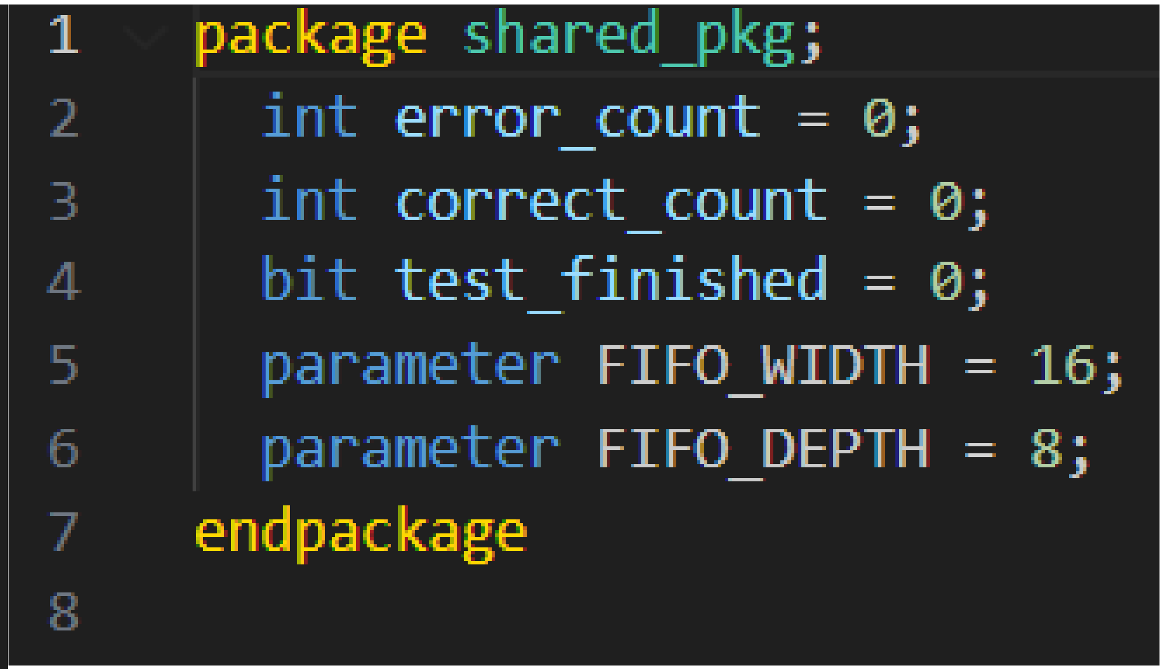


LABEL	DESIGN REQUIREMENT DESCRIPTION	STIMULUS GENERATION	FUNCTIONAL COVERAGE	FUNCTIONALITY CHECK
FIFO_RST	When reset is activated all pointers must equal to the initial state	Directed at the start of the tb then reset is randomized by 5% being active	-	Checking every pointer value by assertions
FIFO_WR	Checking for writing condition if(wr_en&&!full) data should be written on FIFO and at the same time wr_ack =1	Randomized with constraints of wr_en is on 70% of time	a cross coverage is combining all possiblities of wr_en ,wr_ack and rd_en	checking the output value by using a refrence model constructed in check_result task and wr_ack is being checked by assertions
FIFO_RD	checking for read condition if(rd_en&&!empty) data should be passed to data_out	Randomized with constraints of rd_en is on 30% of time	a cross coverage is combining all possiblities of wr_en ,empty and rd_en	a refrence model is used to check the value of data_out and empty is being checked by assertions
FIFO_OVERFLOW	Checking for overflow condition when wr_en is 1 and full is 1 overflow must be asserted	randomizing till getting overflow also after the randomization it was directed to ensure wr_en is 1 for 8 cycles to check for overflow	a cross coverage is used to combining all possiblities of wr_en,overflow,rd_en	a refrence model to check data_out also overflow is checked by assertions
FIFO_UNDERFLOW	Checking for underflow condition when rd_en is 1 and empty is 1 underflow must be asserted	randomizing till getting overflow also after the randomization it was directed to ensure rd_en is 1 for 8 cycles to check for underflow	a cross coverage is used to combining all possiblities of wr_en,underflow,rd_en	a refrence model to check data_out also underflow is checked by assertions
FIFO_FULL	checking for full condition when count=8 this means it wrote 8 times equal to fifo_depth then Full is asserted	randomizing till getting FULL also after the randomization it was directed to ensure wr_en is 1 for 8 cycles	a cross coverage is used to combining all possiblities of wr_en,full,rd_en	a refrence model to check data_out also full is checked by assertions
FIFO_EMPTY	checking for empty condition when count=0 this means it wrote 0 times then empty is asserted	randomizing till getting EMPTY also after the randomization it was directed after reseting to ensure wr_en is 0 and rd_en=1 ensure no data was written	a cross coverage is used to combining all possiblities of wr_en,empty,rd_en	a refrence model to check data_out also empty is checked by assertions
FIFO_ALMOSTFULL	checking for almostfull condition when count=7 this means it wrote 7 times then almostfull is asserted	randomizing till getting ALMOSTFULL also after the randomization it was directed to ensure wr_en is 1 for 7 cycles	a cross coverage is used to combining all possiblities of wr_en,almostfull,rd_en	a refrence model to check data_out also almostfull is checked by assertions
FIFO_ALMOSTEMPTY	checking for empty condition when count=1 this means it wrote 1 times then empty is asserted	randomizing till getting ALMOSTEMPTY also after the randomization it	a cross coverage is used to combining all possiblities of wr_en,almostempty,rd_en	a refrence model to check data_out also almostempty is checked by assertions

FIFO INTERFACE AND SHARED_PKG

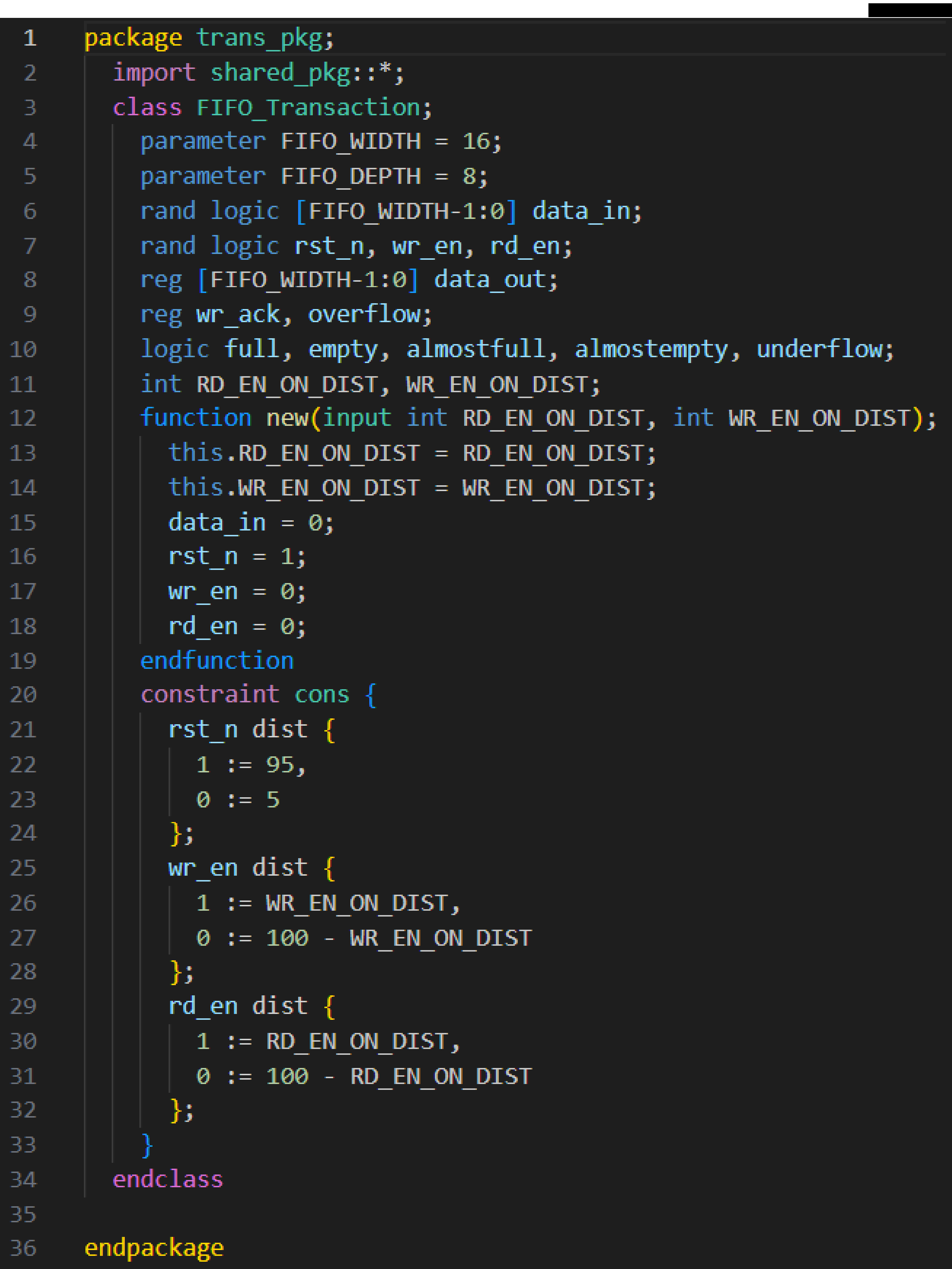


```
interface FIFO_IF (
    input bit clk
);
  parameter FIFO_WIDTH = 16;
  parameter FIFO_DEPTH = 8;
  bit [FIFO_WIDTH-1:0] data_in;
  logic rst_n, wr_en, rd_en;
  reg [FIFO_WIDTH-1:0] data_out;
  reg wr_ack, overflow;
  logic full, empty, almostfull, almostempty, underflow;
  modport TEST(
      output data_in, clk, rst_n, wr_en, rd_en,
      input data_out, wr_ack, overflow, full, empty, almostfull, almostempty, underflow
  );
  modport MON(
      input data_in,clk,rst_n,wr_en,rd_en,
   data_out,wr_ack,overflow,full,empty,almostfull,almostempty,underflow
  );
endinterface
```

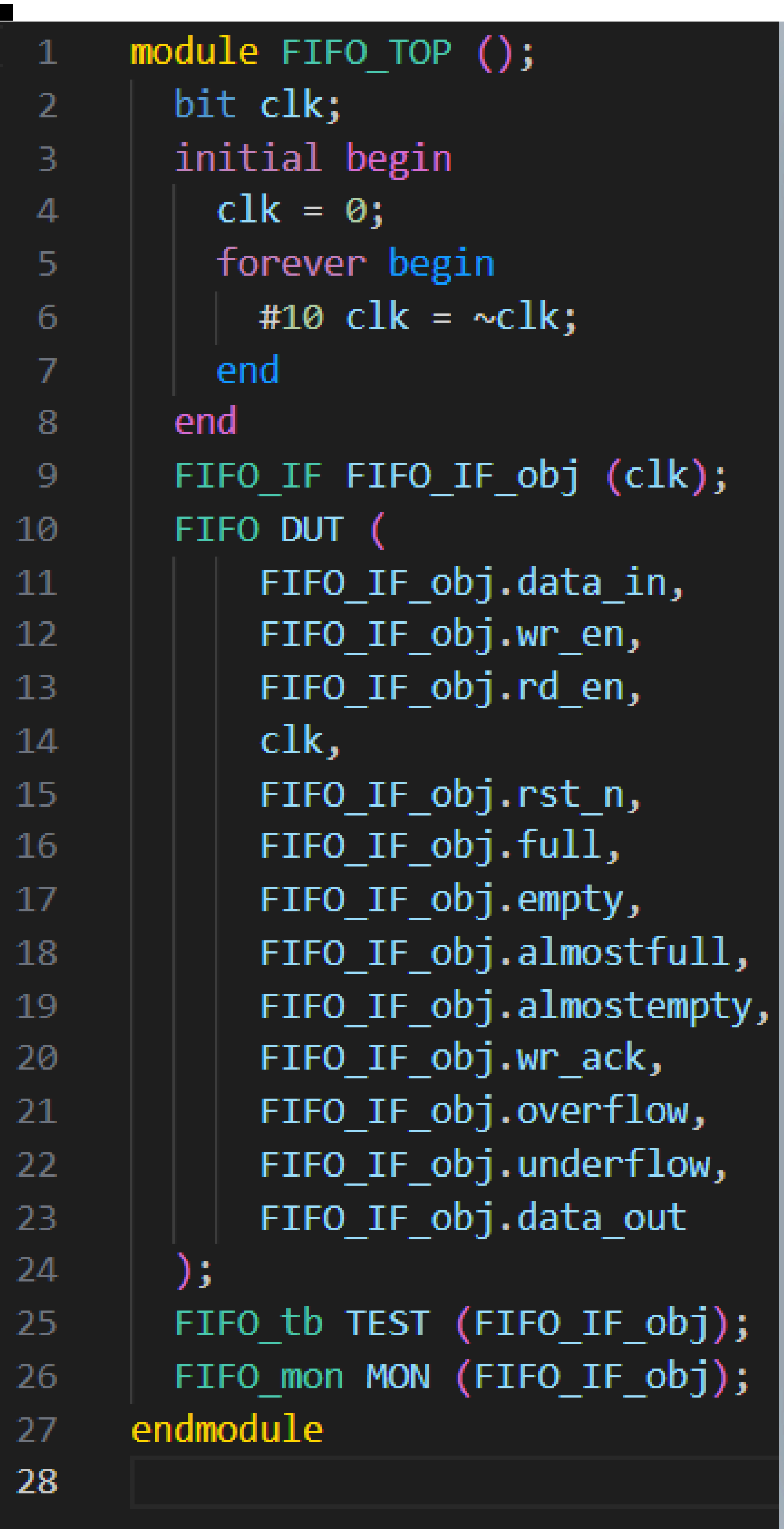


```
package shared_pkg;
  int error_count = 0;
  int correct_count = 0;
  bit test_finished = 0;
  parameter FIFO_WIDTH = 16;
  parameter FIFO_DEPTH = 8;
endpackage
```

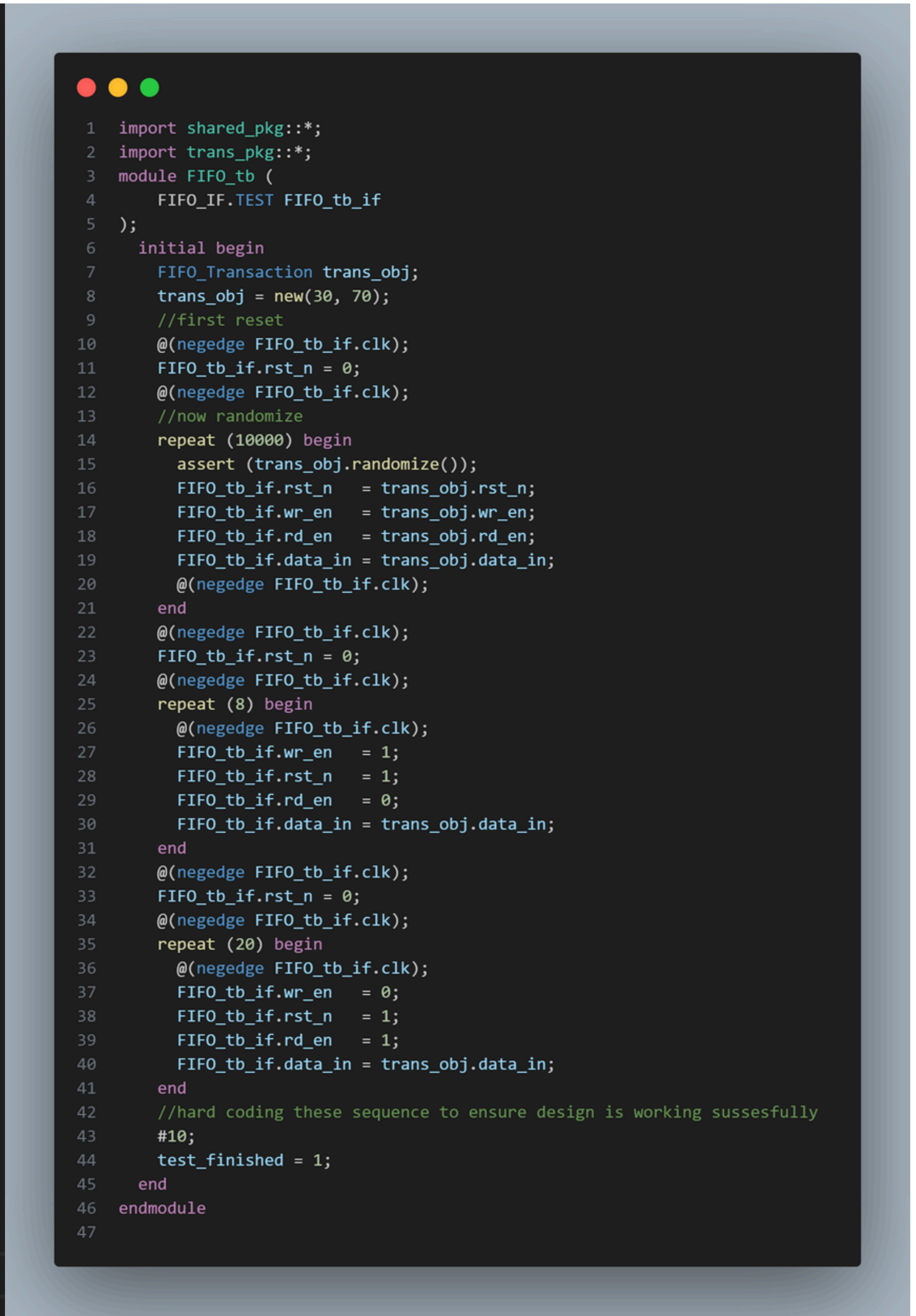
FIFO_TRANSACTION
TOP_MODULE AND TB



```systemverilog
package trans_pkg;
  import shared_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;
    reg [FIFO_WIDTH-1:0] data_out;
    reg wr_ack, overflow;
    logic full, empty, almostfull, almostempty, underflow;
    int RD_EN_ON_DIST, WR_EN_ON_DIST;
    function new(input int RD_EN_ON_DIST, int WR_EN_ON_DIST);
      this.RD_EN_ON_DIST = RD_EN_ON_DIST;
      this.WR_EN_ON_DIST = WR_EN_ON_DIST;
      data_in = 0;
      rst_n = 1;
      wr_en = 0;
      rd_en = 0;
    endfunction
    constraint cons {
      rst_n dist {
        1 := 95,
        0 := 5
      };
      wr_en dist {
        1 := WR_EN_ON_DIST,
        0 := 100 - WR_EN_ON_DIST
      };
      rd_en dist {
        1 := RD_EN_ON_DIST,
        0 := 100 - RD_EN_ON_DIST
      };
    }
  endclass

endpackage
```

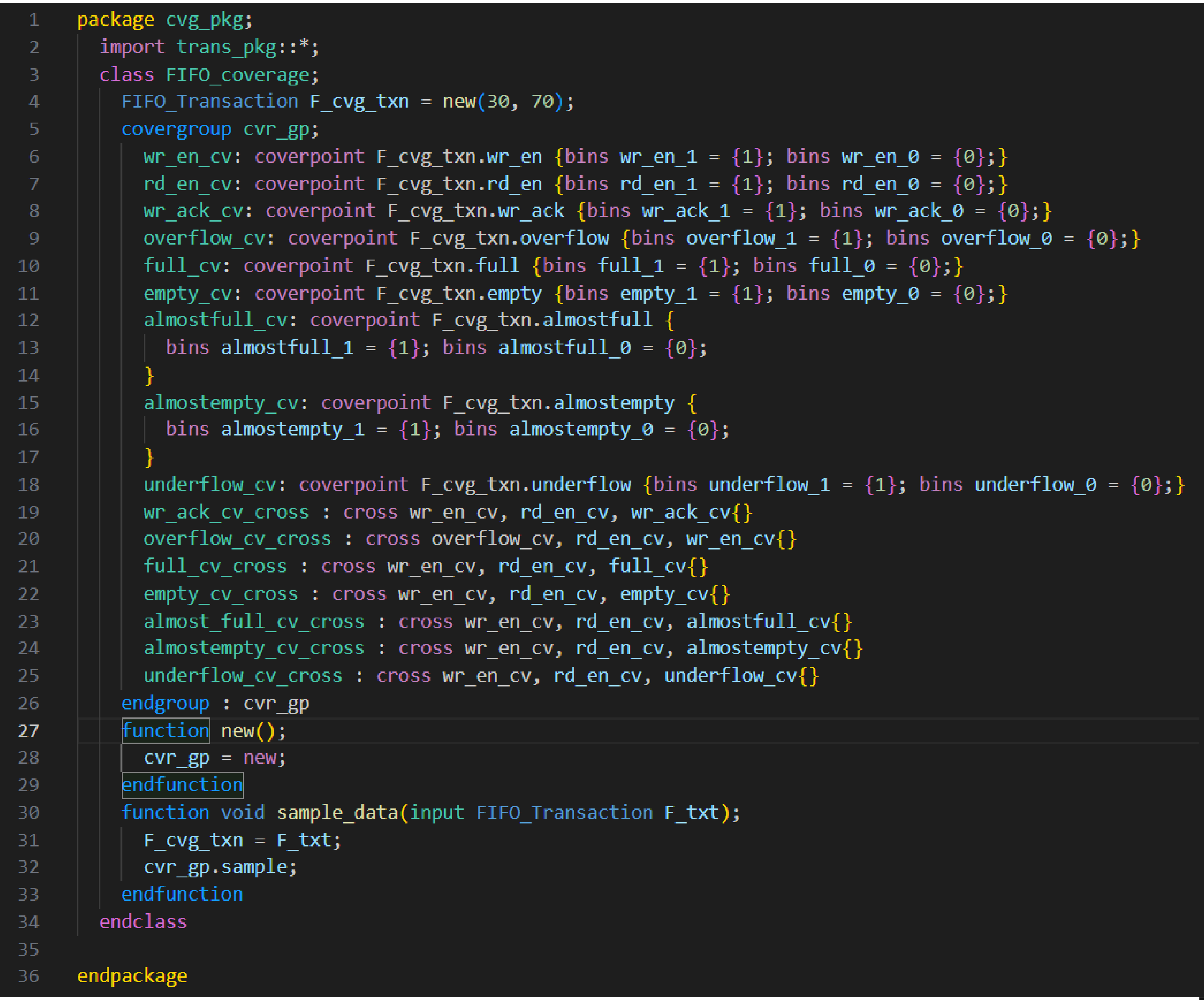


```systemverilog
module FIFO_TOP ();
  bit clk;
  initial begin
    clk = 0;
    forever begin
      #10 clk = ~clk;
    end
  end
  FIFO_IF FIFO_IF_obj (clk);
  FIFO DUT (
      FIFO_IF_obj.data_in,
      FIFO_IF_obj.wr_en,
      FIFO_IF_obj.rd_en,
      clk,
      FIFO_IF_obj.rst_n,
      FIFO_IF_obj.full,
      FIFO_IF_obj.empty,
      FIFO_IF_obj.almostfull,
      FIFO_IF_obj.almostempty,
      FIFO_IF_obj.wr_ack,
      FIFO_IF_obj.overflow,
      FIFO_IF_obj.underflow,
      FIFO_IF_obj.data_out
  );
  FIFO_tb TEST (FIFO_IF_obj);
  FIFO_mon MON (FIFO_IF_obj);
endmodule
```

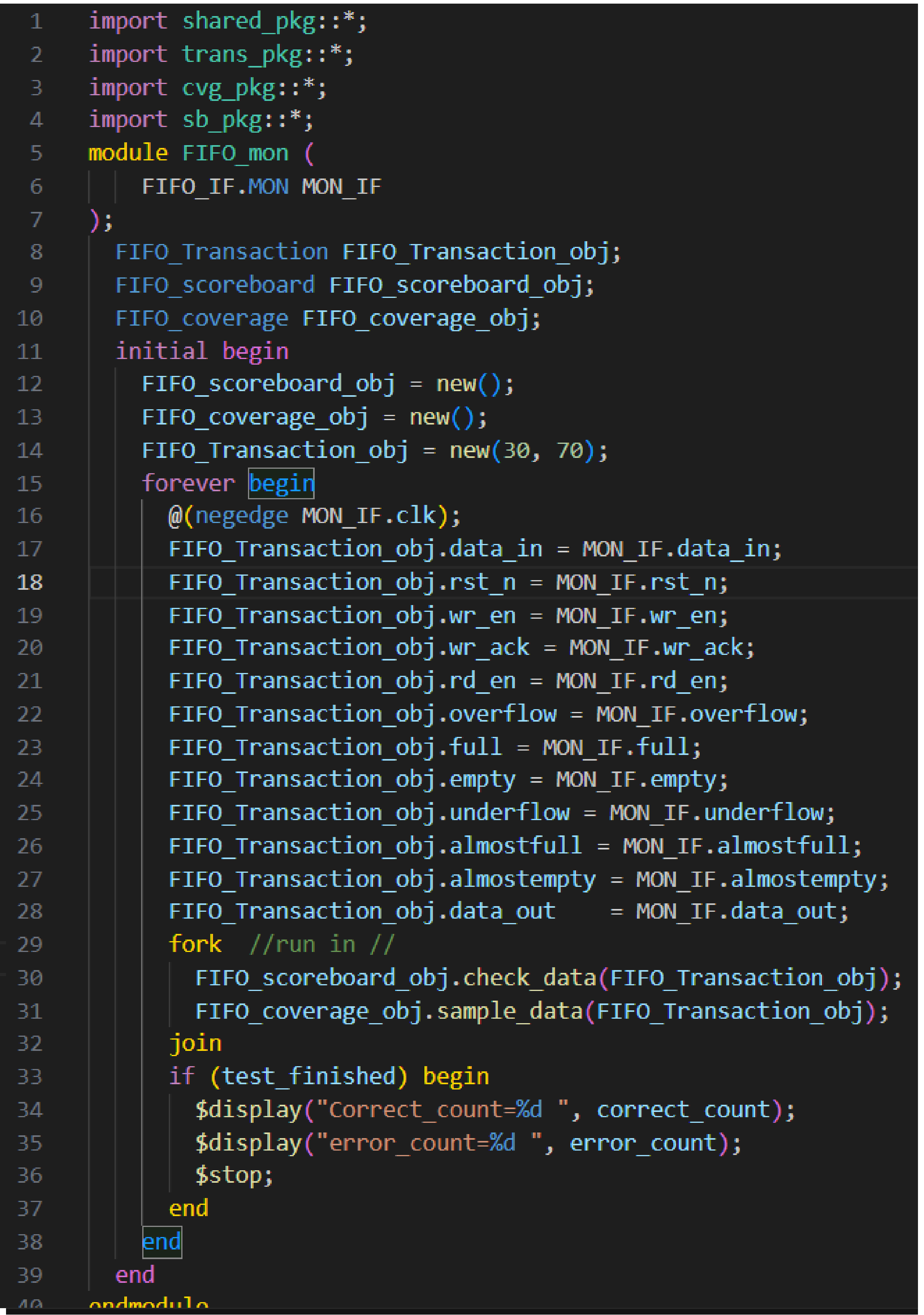


```systemverilog
import shared_pkg::*;
import trans_pkg::*;
module FIFO_tb (
    FIFO_IF.TEST FIFO_tb_if
);
  initial begin
    FIFO_Transaction trans_obj;
    trans_obj = new(30, 70);
    //first reset
    @(negedge FIFO_tb_if.clk);
    FIFO_tb_if.rst_n = 0;
    @(negedge FIFO_tb_if.clk);
    //now randomize
    repeat (10000) begin
      assert (trans_obj.randomize());
      FIFO_tb_if.rst_n   = trans_obj.rst_n;
      FIFO_tb_if.wr_en   = trans_obj.wr_en;
      FIFO_tb_if.rd_en   = trans_obj.rd_en;
      FIFO_tb_if.data_in = trans_obj.data_in;
      @(negedge FIFO_tb_if.clk);
    end
    @(negedge FIFO_tb_if.clk);
    FIFO_tb_if.rst_n = 0;
    @(negedge FIFO_tb_if.clk);
    repeat (8) begin
      @(negedge FIFO_tb_if.clk);
      FIFO_tb_if.wr_en   = 1;
      FIFO_tb_if.rst_n   = 1;
      FIFO_tb_if.rd_en   = 0;
      FIFO_tb_if.data_in = trans_obj.data_in;
    end
    @(negedge FIFO_tb_if.clk);
    FIFO_tb_if.rst_n = 0;
    @(negedge FIFO_tb_if.clk);
    repeat (20) begin
      @(negedge FIFO_tb_if.clk);
      FIFO_tb_if.wr_en   = 0;
      FIFO_tb_if.rst_n   = 1;
      FIFO_tb_if.rd_en   = 1;
      FIFO_tb_if.data_in = trans_obj.data_in;
    end
    //hard coding these sequence to ensure design is working sussesfully
    #10;
    test_finished = 1;
  end
endmodule
```

FIFO COVERAGE AND MONITOR

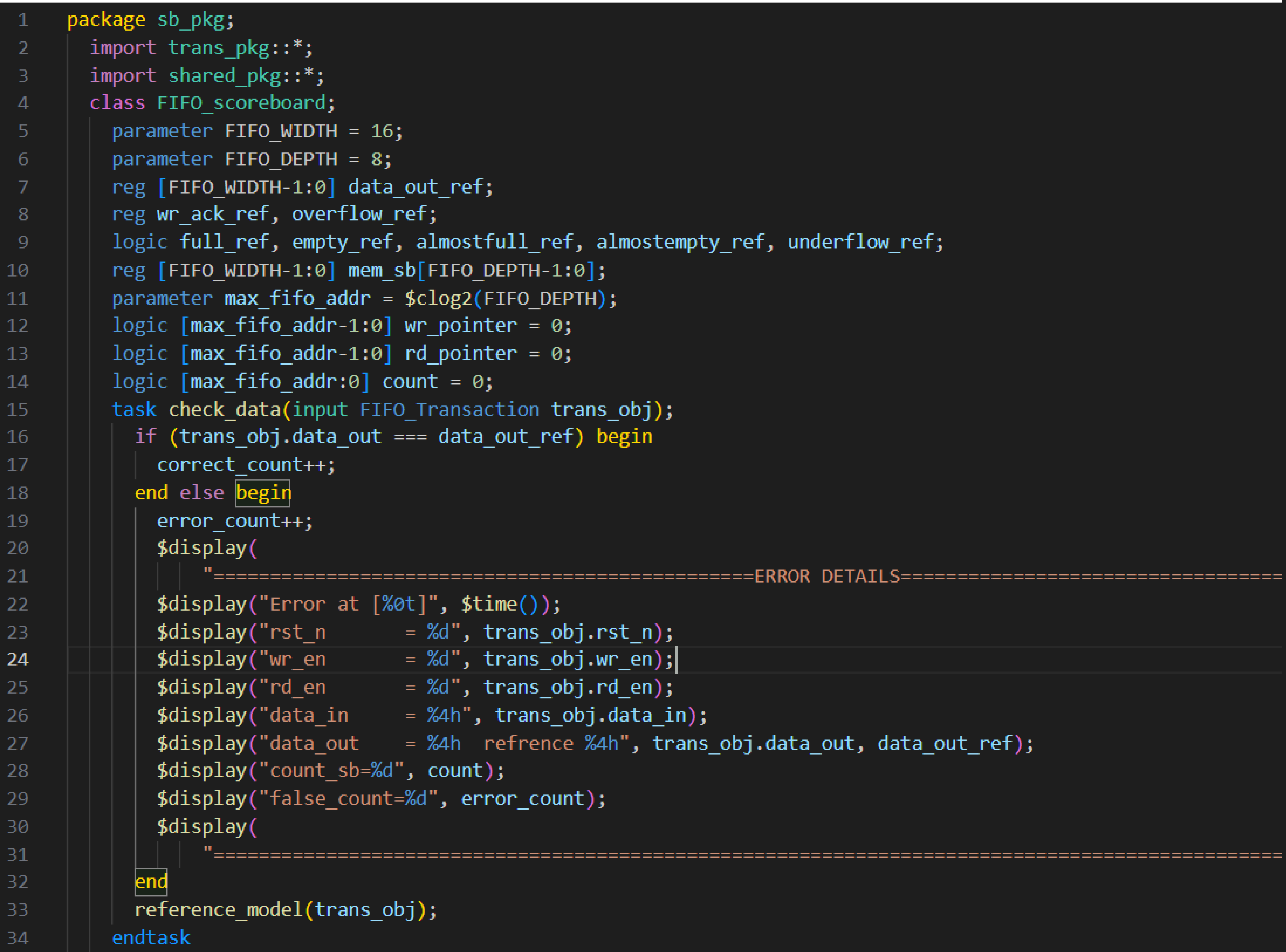


```
package cvg_pkg;
  import trans_pkg::*;
  class FIFO_coverage;
    FIFO_Transaction F_cvg_txn = new(30, 70);
    covergroup cvr_gp;
      wr_en_cv: coverpoint F_cvg_txn.wr_en {bins wr_en_1 = {1}; bins wr_en_0 = {0};}
      rd_en_cv: coverpoint F_cvg_txn.rd_en {bins rd_en_1 = {1}; bins rd_en_0 = {0};}
      wr_ack_cv: coverpoint F_cvg_txn.wr_ack {bins wr_ack_1 = {1}; bins wr_ack_0 = {0};}
      overflow_cv: coverpoint F_cvg_txn.overflow {bins overflow_1 = {1}; bins overflow_0 = {0};}
      full_cv: coverpoint F_cvg_txn.full {bins full_1 = {1}; bins full_0 = {0};}
      empty_cv: coverpoint F_cvg_txn.empty {bins empty_1 = {1}; bins empty_0 = {0};}
      almostfull_cv: coverpoint F_cvg_txn.almostfull {
        bins almostfull_1 = {1}; bins almostfull_0 = {0};
      }
      almostempty_cv: coverpoint F_cvg_txn.almostempty {
        bins almostempty_1 = {1}; bins almostempty_0 = {0};
      }
      underflow_cv: coverpoint F_cvg_txn.underflow {bins underflow_1 = {1}; bins underflow_0 = {0};}
      wr_ack_cv_cross : cross wr_en_cv, rd_en_cv, wr_ack_cv{}
      overflow_cv_cross : cross overflow_cv, rd_en_cv, wr_en_cv{}
      full_cv_cross : cross wr_en_cv, rd_en_cv, full_cv{}
      empty_cv_cross : cross wr_en_cv, rd_en_cv, empty_cv{}
      almost_full_cv_cross : cross wr_en_cv, rd_en_cv, almostfull_cv{}
      almostempty_cv_cross : cross wr_en_cv, rd_en_cv, almostempty_cv{}
      underflow_cv_cross : cross wr_en_cv, rd_en_cv, underflow_cv{}
    endgroup : cvr_gp
    function new();
      cvr_gp = new;
    endfunction
    function void sample_data(input FIFO_Transaction F_txt);
      F_cvg_txn = F_txt;
      cvr_gp.sample;
    endfunction
  endclass

endpackage
```

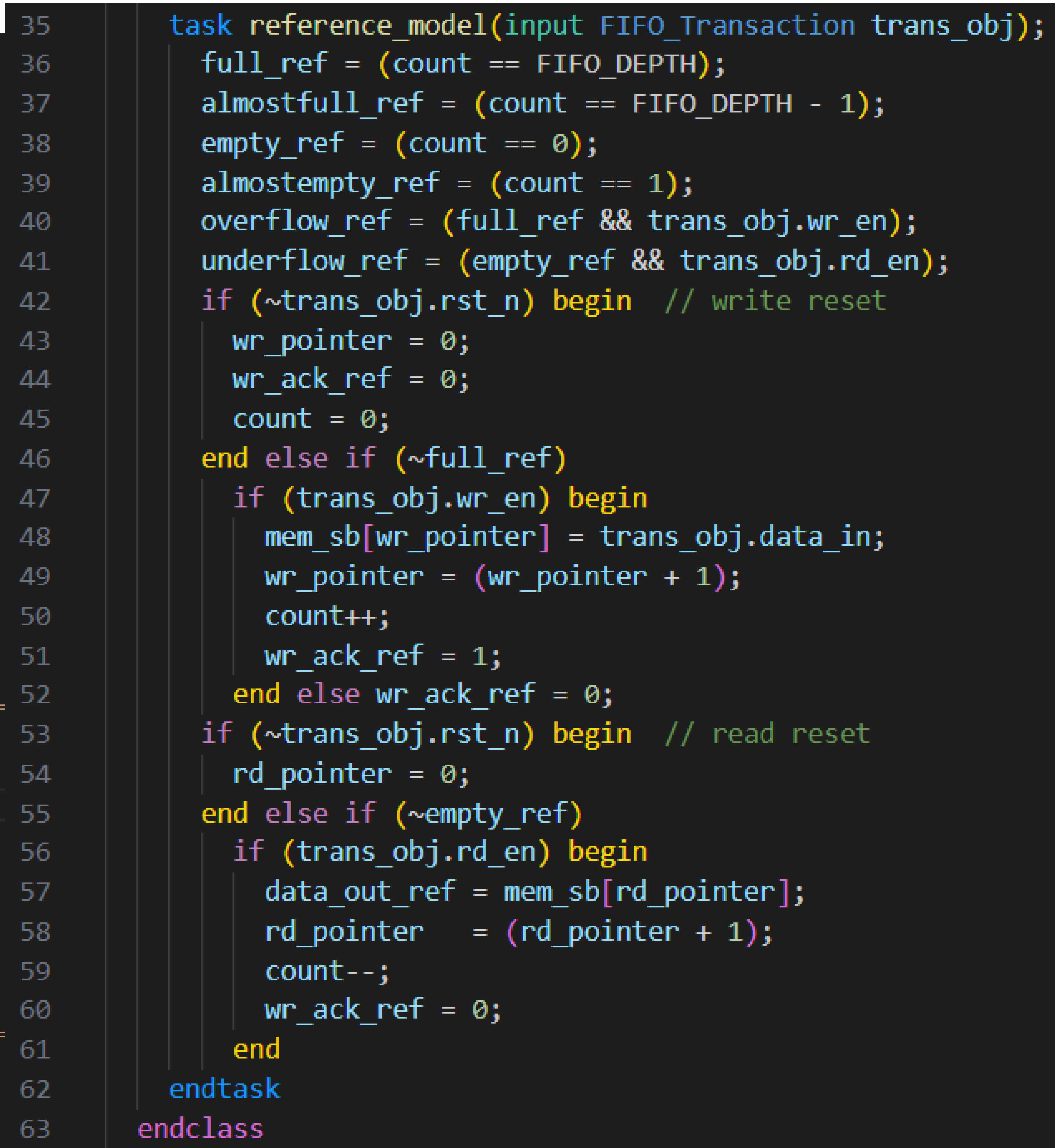


```
import shared_pkg::*;
import trans_pkg::*;
import cvg_pkg::*;
import sb_pkg::*;
module FIFO_mon (
    FIFO_IF.MON MON_IF
);
  FIFO_Transaction FIFO_Transaction_obj;
  FIFO_scoreboard FIFO_scoreboard_obj;
  FIFO_coverage FIFO_coverage_obj;
  initial begin
    FIFO_scoreboard_obj = new();
    FIFO_coverage_obj = new();
    FIFO_Transaction_obj = new(30, 70);
    forever begin
      @(negedge MON_IF.clk);
      FIFO_Transaction_obj.data_in = MON_IF.data_in;
      FIFO_Transaction_obj.rst_n = MON_IF.rst_n;
      FIFO_Transaction_obj.wr_en = MON_IF.wr_en;
      FIFO_Transaction_obj.wr_ack = MON_IF.wr_ack;
      FIFO_Transaction_obj.rd_en = MON_IF.rd_en;
      FIFO_Transaction_obj.overflow = MON_IF.overflow;
      FIFO_Transaction_obj.full = MON_IF.full;
      FIFO_Transaction_obj.empty = MON_IF.empty;
      FIFO_Transaction_obj.underflow = MON_IF.underflow;
      FIFO_Transaction_obj.almostfull = MON_IF.almostfull;
      FIFO_Transaction_obj.almostempty = MON_IF.almostempty;
      FIFO_Transaction_obj.data_out     = MON_IF.data_out;
      fork  //run in //
        FIFO_scoreboard_obj.check_data(FIFO_Transaction_obj);
        FIFO_coverage_obj.sample_data(FIFO_Transaction_obj);
      join
      if (test_finished) begin
        $display("Correct_count=%d ", correct_count);
        $display("error_count=%d ", error_count);
        $stop;
      end
    end
  end
endmodule
```

FIFO_SB

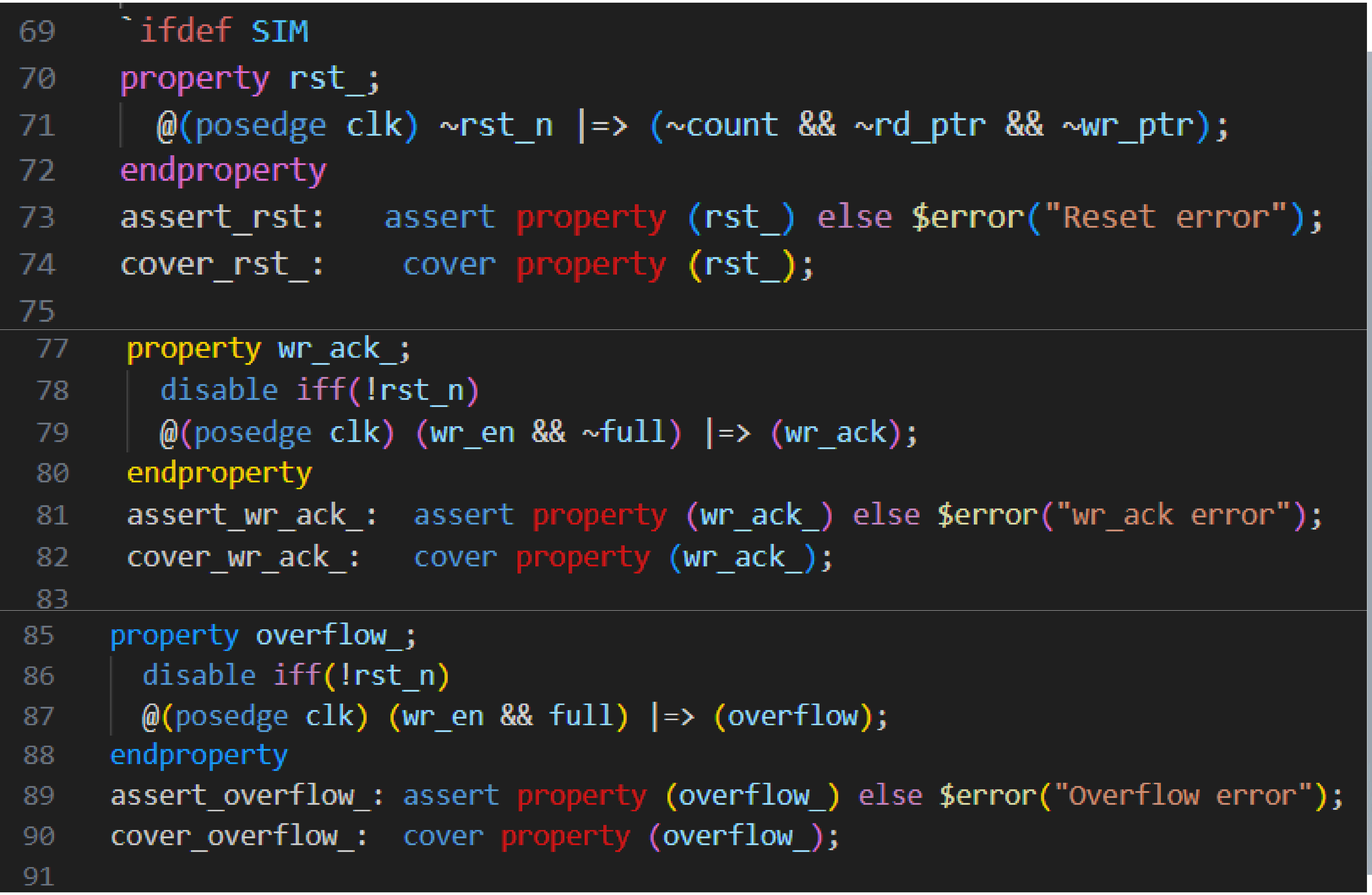


```
package sb_pkg;
  import trans_pkg::*;
  import shared_pkg::*;
  class FIFO_scoreboard;
    parameter FIFO_WIDTH = 16;
    parameter FIFO_DEPTH = 8;
    reg [FIFO_WIDTH-1:0] data_out_ref;
    reg wr_ack_ref, overflow_ref;
    logic full_ref, empty_ref, almostfull_ref, almostempty_ref, underflow_ref;
    reg [FIFO_WIDTH-1:0] mem_sb[FIFO_DEPTH-1:0];
    parameter max_fifo_addr = $clog2(FIFO_DEPTH);
    logic [max_fifo_addr-1:0] wr_pointer = 0;
    logic [max_fifo_addr-1:0] rd_pointer = 0;
    logic [max_fifo_addr:0] count = 0;
    task check_data(input FIFO_Transaction trans_obj);
      if (trans_obj.data_out === data_out_ref) begin
        correct_count++;
      end else begin
        error_count++;
        $display(
            "================================================ERROR DETAILS=================================
        $display("Error at [%0t]", $time());
        $display("rst_n       = %d", trans_obj.rst_n);
        $display("wr_en       = %d", trans_obj.wr_en);
        $display("rd_en       = %d", trans_obj.rd_en);
        $display("data_in     = %4h", trans_obj.data_in);
        $display("data_out    = %4h  refrence %4h", trans_obj.data_out, data_out_ref);
        $display("count_sb=%d", count);
        $display("false_count=%d", error_count);
        $display(
            "==============================================================================================
      end
      reference_model(trans_obj);
    endtask
```

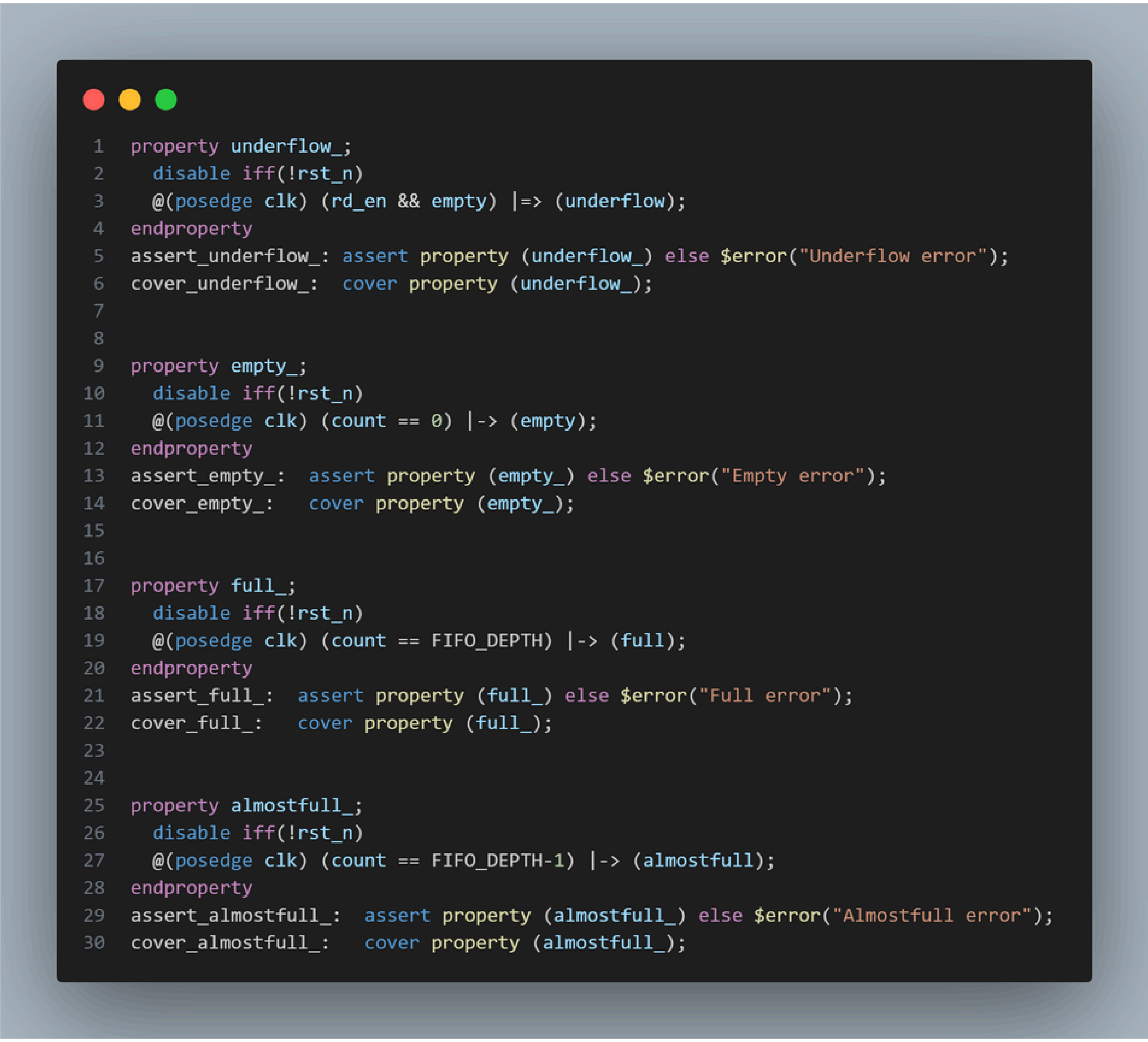


```
    task reference_model(input FIFO_Transaction trans_obj);
      full_ref = (count == FIFO_DEPTH);
      almostfull_ref = (count == FIFO_DEPTH - 1);
      empty_ref = (count == 0);
      almostempty_ref = (count == 1);
      overflow_ref = (full_ref && trans_obj.wr_en);
      underflow_ref = (empty_ref && trans_obj.rd_en);
      if (~trans_obj.rst_n) begin  // write reset
        wr_pointer = 0;
        wr_ack_ref = 0;
        count = 0;
      end else if (~full_ref)
        if (trans_obj.wr_en) begin
          mem_sb[wr_pointer] = trans_obj.data_in;
          wr_pointer = (wr_pointer + 1);
          count++;
          wr_ack_ref = 1;
        end else wr_ack_ref = 0;
      if (~trans_obj.rst_n) begin  // read reset
        rd_pointer = 0;
      end else if (~empty_ref)
        if (trans_obj.rd_en) begin
          data_out_ref = mem_sb[rd_pointer];
          rd_pointer   = (rd_pointer + 1);
          count--;
          wr_ack_ref = 0;
        end
    endtask
  endclass
```

FIFO ASSERTIONS

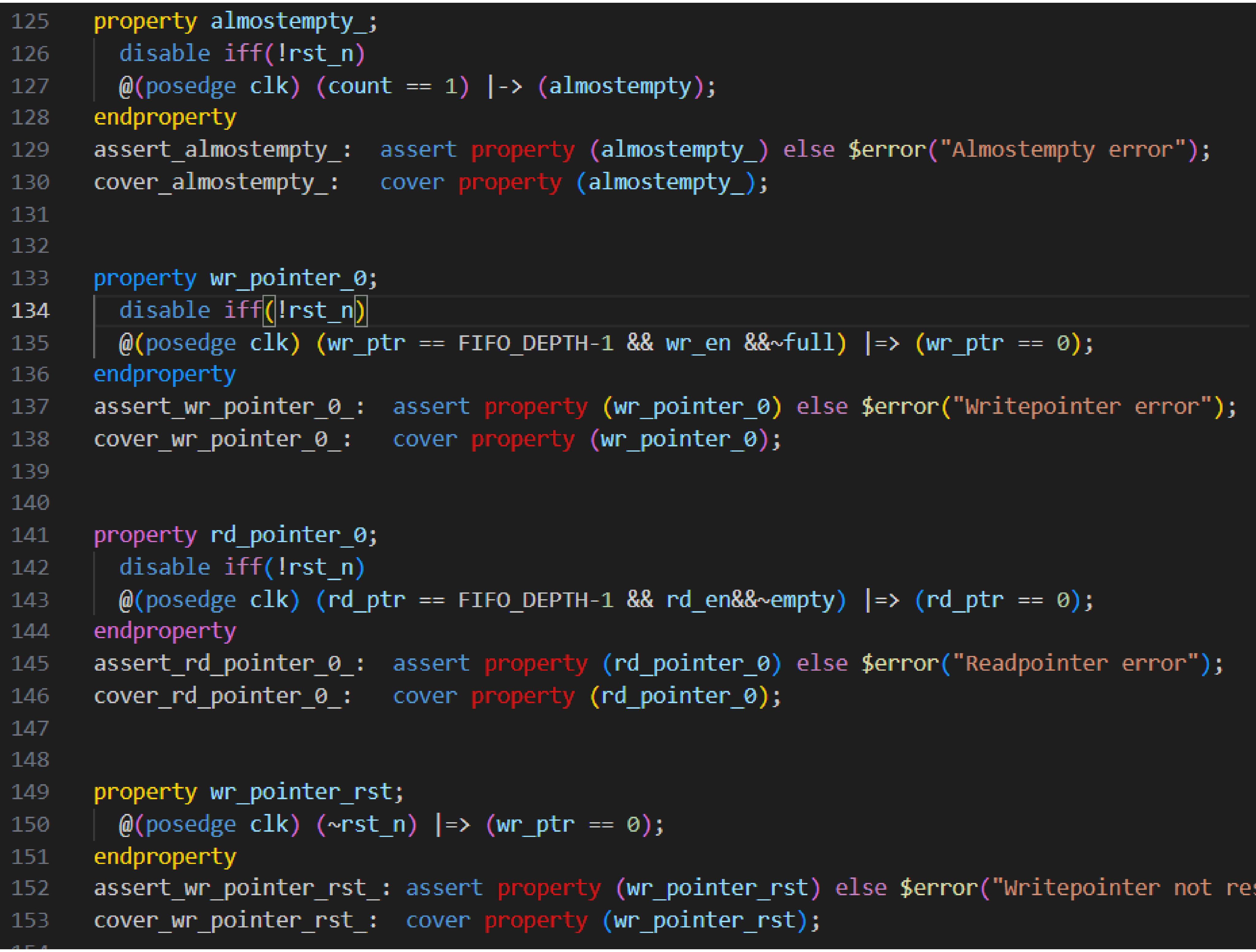


```
`ifdef SIM
property rst_;
  @(posedge clk) ~rst_n |=> (~count && ~rd_ptr && ~wr_ptr);
endproperty
assert_rst:   assert property (rst_) else $error("Reset error");
cover_rst_:    cover property (rst_);

property wr_ack_;
  disable iff(!rst_n)
  @(posedge clk) (wr_en && ~full) |=> (wr_ack);
endproperty
assert_wr_ack_:  assert property (wr_ack_) else $error("wr_ack error");
cover_wr_ack_:   cover property (wr_ack_);

property overflow_;
  disable iff(!rst_n)
  @(posedge clk) (wr_en && full) |=> (overflow);
endproperty
assert_overflow_: assert property (overflow_) else $error("Overflow error");
cover_overflow_:  cover property (overflow_);

```

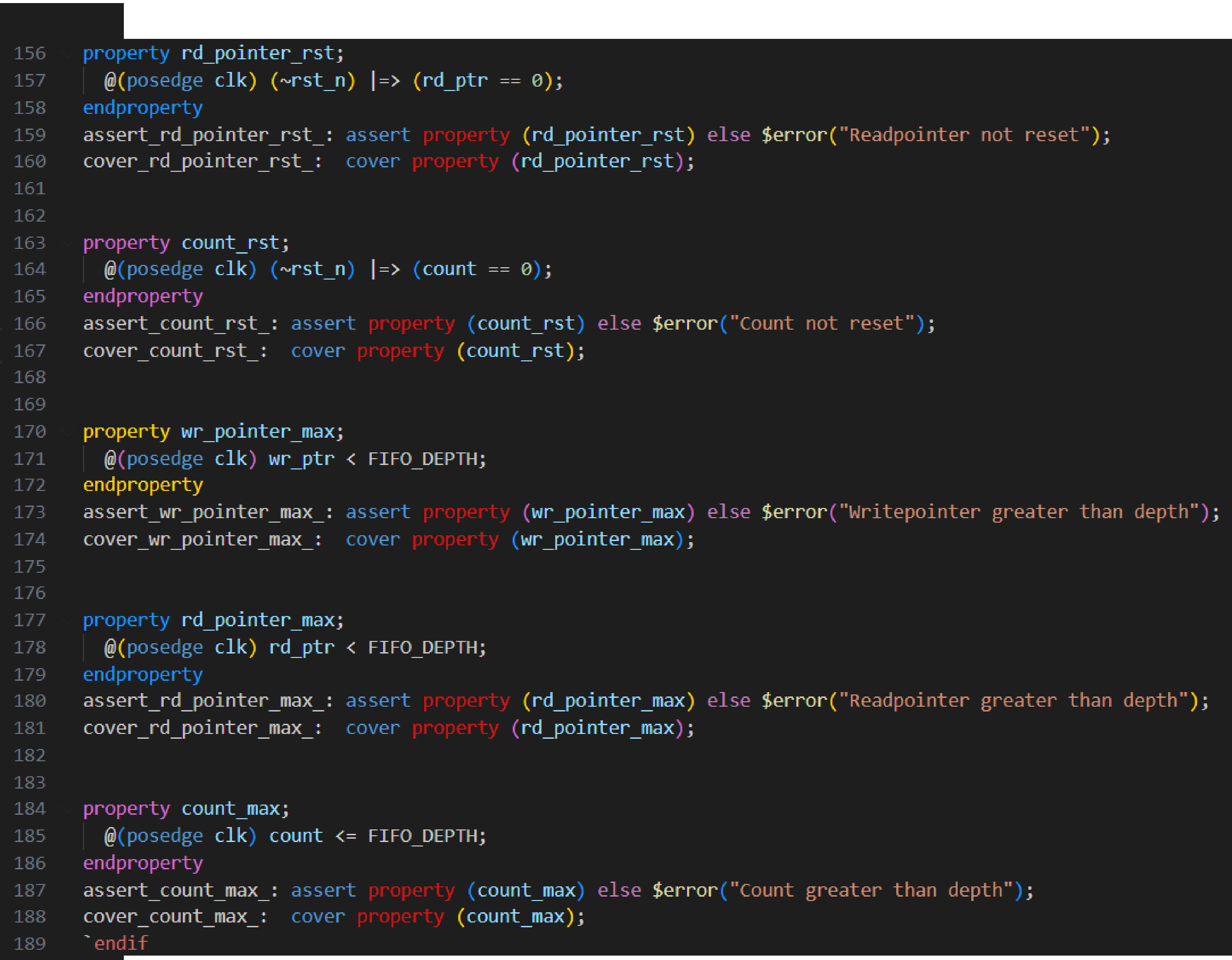


```
property underflow_;
  disable iff(!rst_n)
  @(posedge clk) (rd_en && empty) |=> (underflow);
endproperty
assert_underflow_: assert property (underflow_) else $error("Underflow error");
cover_underflow_:  cover property (underflow_);


property empty_;
  disable iff(!rst_n)
  @(posedge clk) (count == 0) |-> (empty);
endproperty
assert_empty_:  assert property (empty_) else $error("Empty error");
cover_empty_:   cover property (empty_);


property full_;
  disable iff(!rst_n)
  @(posedge clk) (count == FIFO_DEPTH) |-> (full);
endproperty
assert_full_:  assert property (full_) else $error("Full error");
cover_full_:   cover property (full_);


property almostfull_;
  disable iff(!rst_n)
  @(posedge clk) (count == FIFO_DEPTH-1) |-> (almostfull);
endproperty
assert_almostfull_:  assert property (almostfull_) else $error("Almostfull error");
cover_almostfull_:   cover property (almostfull_);
```

FIFO ASSERTIONS

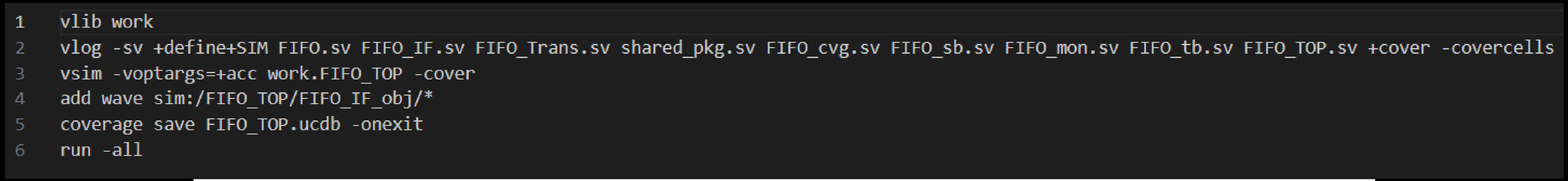


```
property almostempty_;
  disable iff(!rst_n)
  @(posedge clk) (count == 1) |-> (almostempty);
endproperty
assert_almostempty_:  assert property (almostempty_) else $error("Almostempty error");
cover_almostempty_:   cover property (almostempty_);


property wr_pointer_0;
  disable iff(!rst_n)
  @(posedge clk) (wr_ptr == FIFO_DEPTH-1 && wr_en &&~full) |=> (wr_ptr == 0);
endproperty
assert_wr_pointer_0_:  assert property (wr_pointer_0) else $error("Writepointer error");
cover_wr_pointer_0_:   cover property (wr_pointer_0);


property rd_pointer_0;
  disable iff(!rst_n)
  @(posedge clk) (rd_ptr == FIFO_DEPTH-1 && rd_en&&~empty) |=> (rd_ptr == 0);
endproperty
assert_rd_pointer_0_:  assert property (rd_pointer_0) else $error("Readpointer error");
cover_rd_pointer_0_:   cover property (rd_pointer_0);


property wr_pointer_rst;
  @(posedge clk) (~rst_n) |=> (wr_ptr == 0);
endproperty
assert_wr_pointer_rst_: assert property (wr_pointer_rst) else $error("Writepointer not reset");
cover_wr_pointer_rst_:  cover property (wr_pointer_rst);
```

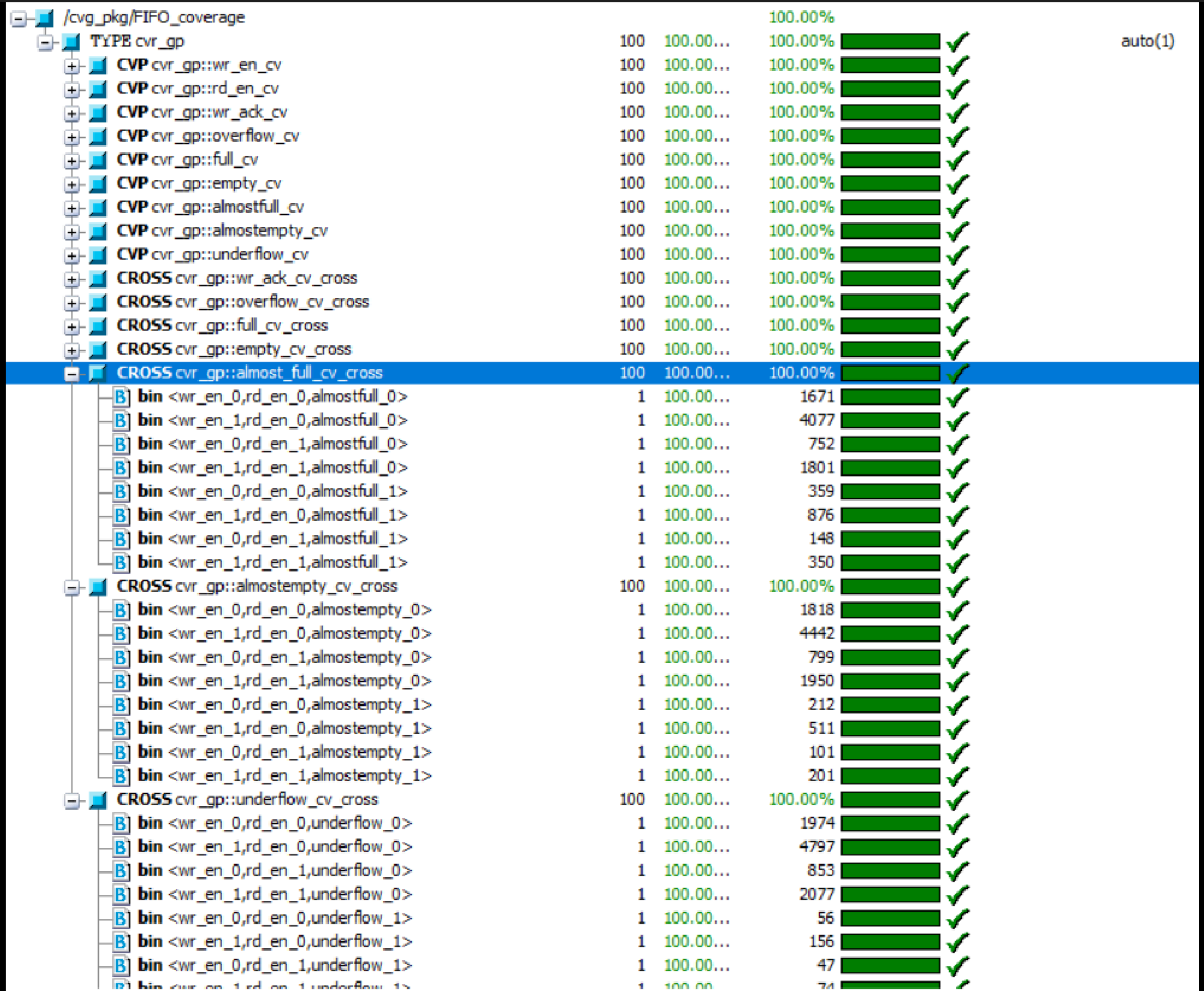


```
property rd_pointer_rst;
  @(posedge clk) (~rst_n) |=> (rd_ptr == 0);
endproperty
assert_rd_pointer_rst_: assert property (rd_pointer_rst) else $error("Readpointer not reset");
cover_rd_pointer_rst_:  cover property (rd_pointer_rst);


property count_rst;
  @(posedge clk) (~rst_n) |=> (count == 0);
endproperty
assert_count_rst_: assert property (count_rst) else $error("Count not reset");
cover_count_rst_:  cover property (count_rst);


property wr_pointer_max;
  @(posedge clk) wr_ptr < FIFO_DEPTH;
endproperty
assert_wr_pointer_max_: assert property (wr_pointer_max) else $error("Writepointer greater than depth");
cover_wr_pointer_max_:  cover property (wr_pointer_max);


property rd_pointer_max;
  @(posedge clk) rd_ptr < FIFO_DEPTH;
endproperty
assert_rd_pointer_max_: assert property (rd_pointer_max) else $error("Readpointer greater than depth");
cover_rd_pointer_max_:  cover property (rd_pointer_max);


property count_max;
  @(posedge clk) count <= FIFO_DEPTH;
endproperty
assert_count_max_: assert property (count_max) else $error("Count greater than depth");
cover_count_max_:  cover property (count_max);
`endif
```

DO FILE AND FUNCTIONAL COVERAGE RESULTS

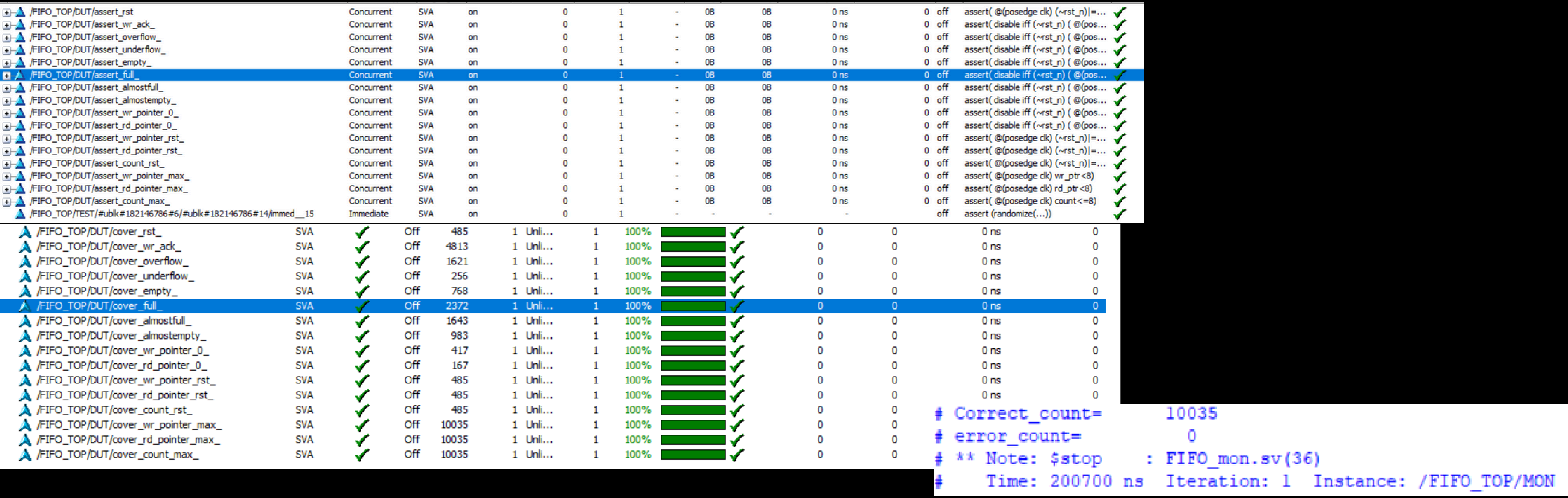


```
vlib work
vlog -sv +define+SIM FIFO.sv FIFO_IF.sv FIFO_Trans.sv shared_pkg.sv FIFO_cvg.sv FIFO_sb.sv FIFO_mon.sv FIFO_tb.sv FIFO_TOP.sv +cover -covercells
vsim -voptargs=+acc work.FIFO_TOP -cover
add wave sim:/FIFO_TOP/FIFO_IF_obj/*
coverage save FIFO_TOP.ucdb -onexit
run -all
```



```
/cvg_pkg/FIFO_coverage                                           100.00%
  TYPE cvr_gp                                   100  100.00...   100.00%  ✓   auto(1)
    CVP cvr_gp::wr_en_cv                        100  100.00...   100.00%  ✓
    CVP cvr_gp::rd_en_cv                        100  100.00...   100.00%  ✓
    CVP cvr_gp::wr_ack_cv                       100  100.00...   100.00%  ✓
    CVP cvr_gp::overflow_cv                     100  100.00...   100.00%  ✓
    CVP cvr_gp::full_cv                         100  100.00...   100.00%  ✓
    CVP cvr_gp::empty_cv                        100  100.00...   100.00%  ✓
    CVP cvr_gp::almostfull_cv                   100  100.00...   100.00%  ✓
    CVP cvr_gp::almostempty_cv                  100  100.00...   100.00%  ✓
    CVP cvr_gp::underflow_cv                    100  100.00...   100.00%  ✓
    CROSS cvr_gp::wr_ack_cv_cross               100  100.00...   100.00%  ✓
    CROSS cvr_gp::overflow_cv_cross             100  100.00...   100.00%  ✓
    CROSS cvr_gp::full_cv_cross                 100  100.00...   100.00%  ✓
    CROSS cvr_gp::empty_cv_cross                100  100.00...   100.00%  ✓
    CROSS cvr_gp::almost_full_cv_cross          100  100.00...   100.00%  ✓
      B] bin <wr_en_0,rd_en_0,almostfull_0>       1  100.00...      1671  ✓
      B] bin <wr_en_1,rd_en_0,almostfull_0>       1  100.00...      4077  ✓
      B] bin <wr_en_0,rd_en_1,almostfull_0>       1  100.00...       752  ✓
      B] bin <wr_en_1,rd_en_1,almostfull_0>       1  100.00...      1801  ✓
      B] bin <wr_en_0,rd_en_0,almostfull_1>       1  100.00...       359  ✓
      B] bin <wr_en_1,rd_en_0,almostfull_1>       1  100.00...       876  ✓
      B] bin <wr_en_0,rd_en_1,almostfull_1>       1  100.00...       148  ✓
      B] bin <wr_en_1,rd_en_1,almostfull_1>       1  100.00...       350  ✓
    CROSS cvr_gp::almostempty_cv_cross          100  100.00...   100.00%  ✓
      B] bin <wr_en_0,rd_en_0,almostempty_0>      1  100.00...      1818  ✓
      B] bin <wr_en_1,rd_en_0,almostempty_0>      1  100.00...      4442  ✓
      B] bin <wr_en_0,rd_en_1,almostempty_0>      1  100.00...       799  ✓
      B] bin <wr_en_1,rd_en_1,almostempty_0>      1  100.00...      1950  ✓
      B] bin <wr_en_0,rd_en_0,almostempty_1>      1  100.00...       212  ✓
      B] bin <wr_en_1,rd_en_0,almostempty_1>      1  100.00...       511  ✓
      B] bin <wr_en_0,rd_en_1,almostempty_1>      1  100.00...       101  ✓
      B] bin <wr_en_1,rd_en_1,almostempty_1>      1  100.00...       201  ✓
    CROSS cvr_gp::underflow_cv_cross            100  100.00...   100.00%  ✓
      B] bin <wr_en_0,rd_en_0,underflow_0>        1  100.00...      1974  ✓
      B] bin <wr_en_1,rd_en_0,underflow_0>        1  100.00...      4797  ✓
      B] bin <wr_en_0,rd_en_1,underflow_0>        1  100.00...       853  ✓
      B] bin <wr_en_1,rd_en_1,underflow_0>        1  100.00...      2077  ✓
      B] bin <wr_en_0,rd_en_0,underflow_1>        1  100.00...        56  ✓
      B] bin <wr_en_1,rd_en_0,underflow_1>        1  100.00...       156  ✓
      B] bin <wr_en_0,rd_en_1,underflow_1>        1  100.00...        47  ✓
      B] bin <wr_en_1,rd_en_1,underflow_1>        1  100.00...        74  ✓
```

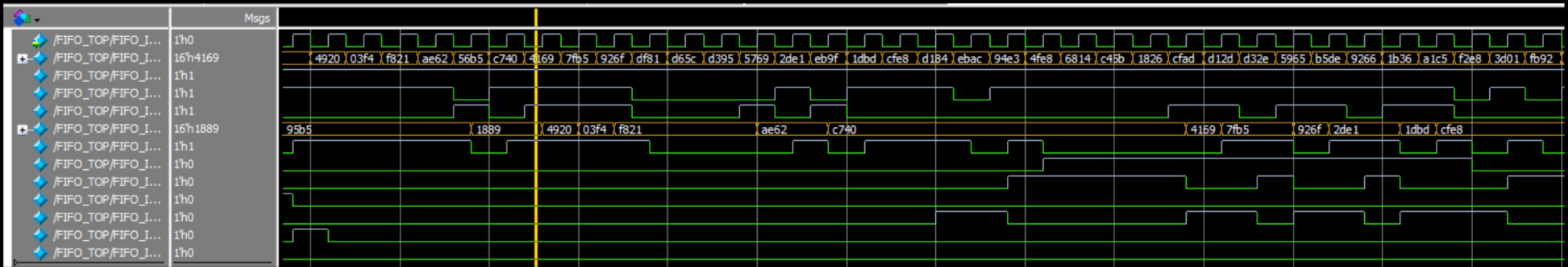
ASSERTIONS RESULTS



/FIFO_TOP/DUT/assert_rst	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) (~rst_n)|=...	✓
/FIFO_TOP/DUT/assert_wr_ack_	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(disable iff (~rst_n) (@(pos...	✓
/FIFO_TOP/DUT/assert_overflow_	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(disable iff (~rst_n) (@(pos...	✓
/FIFO_TOP/DUT/assert_underflow_	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(disable iff (~rst_n) (@(pos...	✓
/FIFO_TOP/DUT/assert_empty_	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(disable iff (~rst_n) (@(pos...	✓
/FIFO_TOP/DUT/assert_full_	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(disable iff (~rst_n) (@(pos...	✓
/FIFO_TOP/DUT/assert_almostfull_	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(disable iff (~rst_n) (@(pos...	✓
/FIFO_TOP/DUT/assert_almostempty_	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(disable iff (~rst_n) (@(pos...	✓
/FIFO_TOP/DUT/assert_wr_pointer_0_	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(disable iff (~rst_n) (@(pos...	✓
/FIFO_TOP/DUT/assert_rd_pointer_0_	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(disable iff (~rst_n) (@(pos...	✓
/FIFO_TOP/DUT/assert_wr_pointer_rst_	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) (~rst_n)|=...	✓
/FIFO_TOP/DUT/assert_rd_pointer_rst_	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) (~rst_n)|=...	✓
/FIFO_TOP/DUT/assert_count_rst_	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) (~rst_n)|=...	✓
/FIFO_TOP/DUT/assert_wr_pointer_max_	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) wr_ptr<8)	✓
/FIFO_TOP/DUT/assert_rd_pointer_max_	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) rd_ptr<8)	✓
/FIFO_TOP/DUT/assert_count_max_	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) count<=8)	✓
/FIFO_TOP/TEST/#ublk#182146786#6/#ublk#182146786#14/immed__15	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓

/FIFO_TOP/DUT/cover_rst_	SVA	✓	Off	485	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/FIFO_TOP/DUT/cover_wr_ack_	SVA	✓	Off	4813	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/FIFO_TOP/DUT/cover_overflow_	SVA	✓	Off	1621	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/FIFO_TOP/DUT/cover_underflow_	SVA	✓	Off	256	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/FIFO_TOP/DUT/cover_empty_	SVA	✓	Off	768	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/FIFO_TOP/DUT/cover_full_	SVA	✓	Off	2372	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/FIFO_TOP/DUT/cover_almostfull_	SVA	✓	Off	1643	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/FIFO_TOP/DUT/cover_almostempty_	SVA	✓	Off	983	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/FIFO_TOP/DUT/cover_wr_pointer_0_	SVA	✓	Off	417	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/FIFO_TOP/DUT/cover_rd_pointer_0_	SVA	✓	Off	167	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/FIFO_TOP/DUT/cover_wr_pointer_rst_	SVA	✓	Off	485	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/FIFO_TOP/DUT/cover_rd_pointer_rst_	SVA	✓	Off	485	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/FIFO_TOP/DUT/cover_count_rst_	SVA	✓	Off	485	1	Unli...	1	100%	🟩	✓	0	0		
/FIFO_TOP/DUT/cover_wr_pointer_max_	SVA	✓	Off	10035	1	Unli...	1	100%	🟩	✓	0	0		
/FIFO_TOP/DUT/cover_rd_pointer_max_	SVA	✓	Off	10035	1	Unli...	1	100%	🟩	✓	0	0		
/FIFO_TOP/DUT/cover_count_max_	SVA	✓	Off	10035	1	Unli...	1	100%	🟩	✓	0	0		

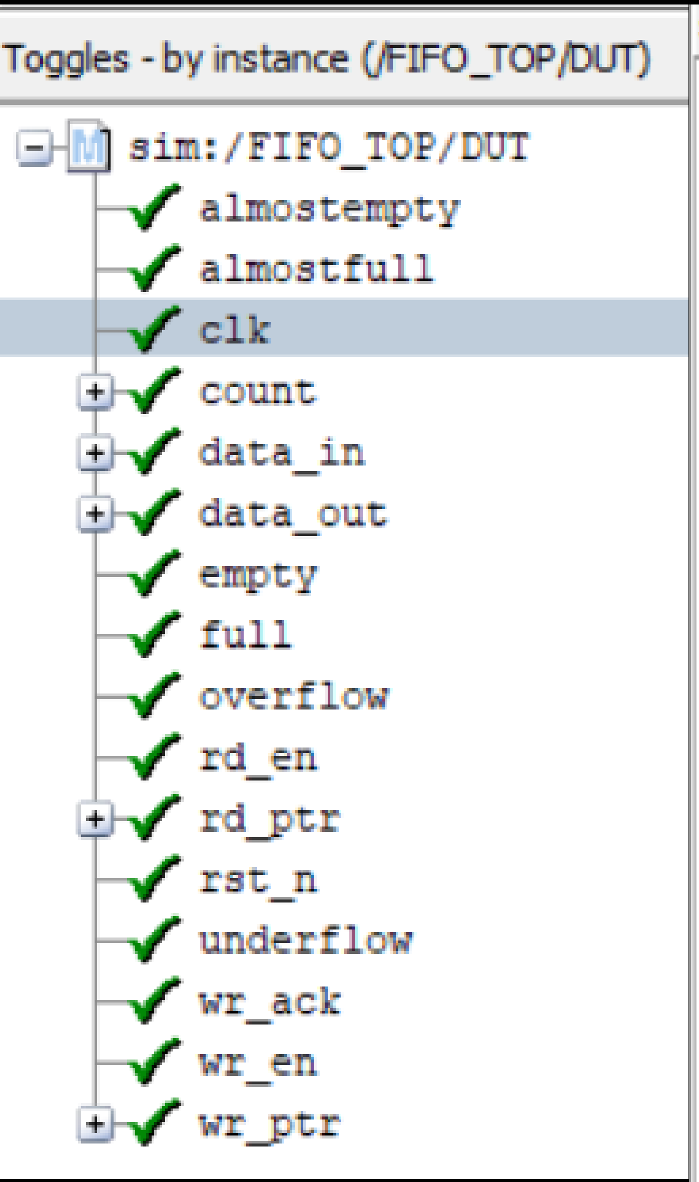
```
# Correct_count=        10035
# error_count=             0
# ** Note: $stop    : FIFO_mon.sv(36)
#    Time: 200700 ns  Iteration: 1  Instance: /FIFO_TOP/MON
```



```
                    Msgs
/FIFO_TOP/FIFO_I... 1'h0
/FIFO_TOP/FIFO_I... 16'h4169
/FIFO_TOP/FIFO_I... 1'h1
/FIFO_TOP/FIFO_I... 1'h1
/FIFO_TOP/FIFO_I... 1'h1
/FIFO_TOP/FIFO_I... 16'h1889
/FIFO_TOP/FIFO_I... 1'h1
/FIFO_TOP/FIFO_I... 1'h0
/FIFO_TOP/FIFO_I... 1'h0
/FIFO_TOP/FIFO_I... 1'h0
/FIFO_TOP/FIFO_I... 1'h0
/FIFO_TOP/FIFO_I... 1'h0
/FIFO_TOP/FIFO_I... 1'h0
```

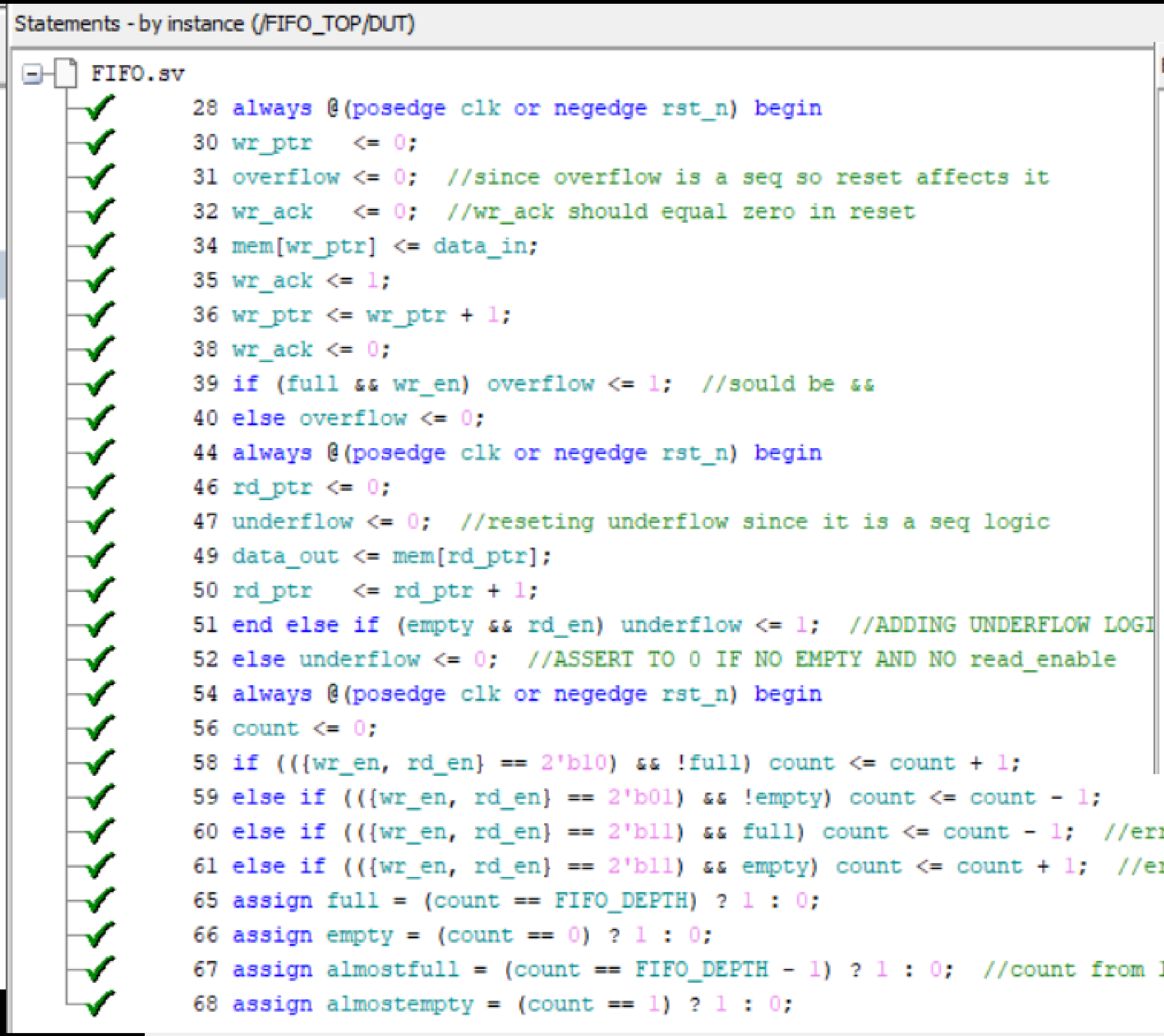
4920 03f4 f821 ae62 56b5 c740 4169 7fb5 926f df81 d65c d395 5769 2de1 eb9f 1dbd cfe8 d184 ebac 94e3 4fe8 6814 c45b 1826 cfad d12d d32e 5965 b5de 9266 1b36 a1c5 f2e8 3d01 fb92
95b5 1889 4920 03f4 f821 ae62 c740 4169 7fb5 926f 2de1 1dbd cfe8

COVERAGE



Toggles - by instance (/FIFO_TOP/DUT)

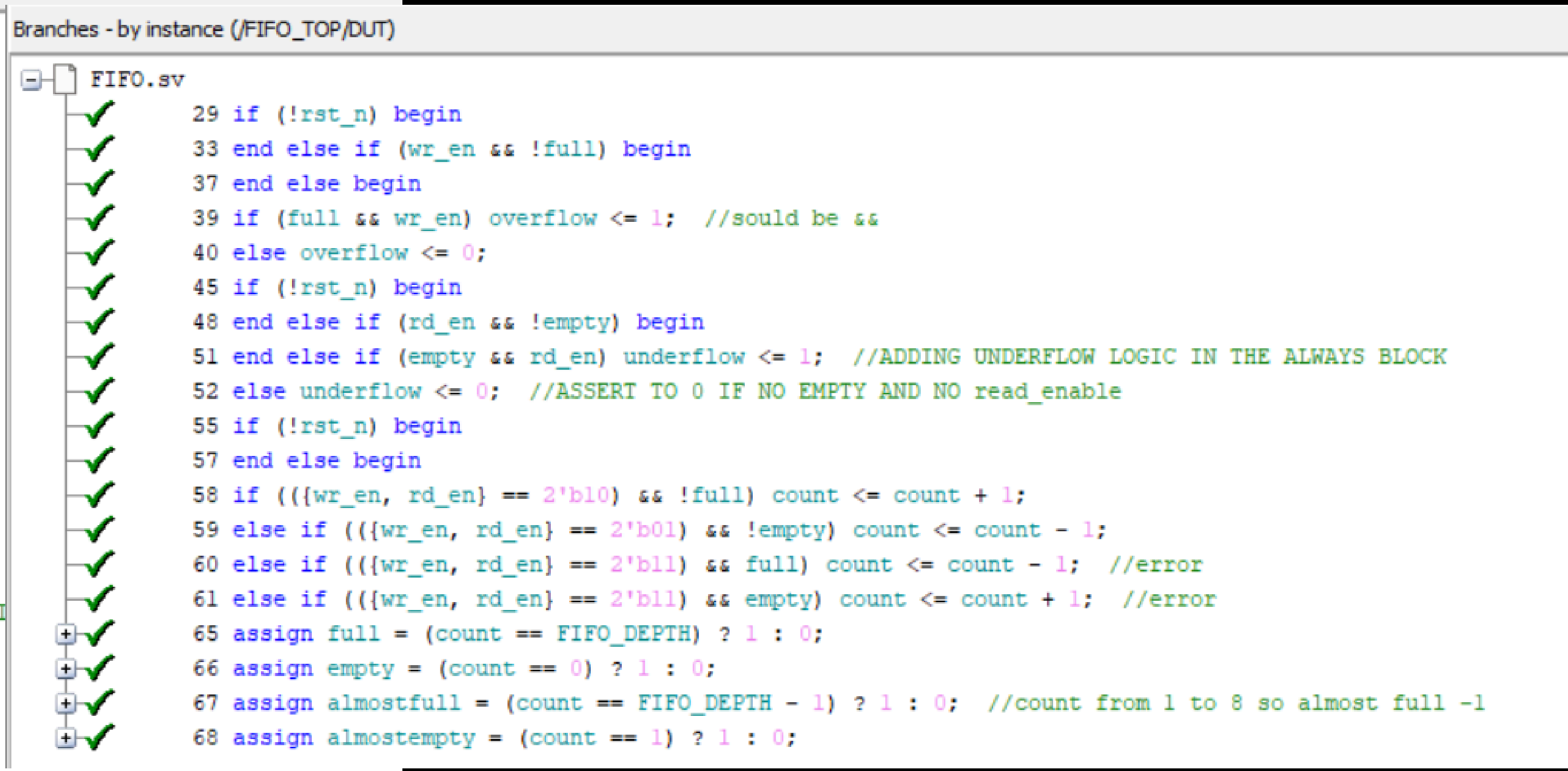
- sim:/FIFO_TOP/DUT
  - ✓ almostempty
  - ✓ almostfull
  - ✓ clk
  - ✓ count
  - ✓ data_in
  - ✓ data_out
  - ✓ empty
  - ✓ full
  - ✓ overflow
  - ✓ rd_en
  - ✓ rd_ptr
  - ✓ rst_n
  - ✓ underflow
  - ✓ wr_ack
  - ✓ wr_en
  - ✓ wr_ptr



Statements - by instance (/FIFO_TOP/DUT)

FIFO.sv

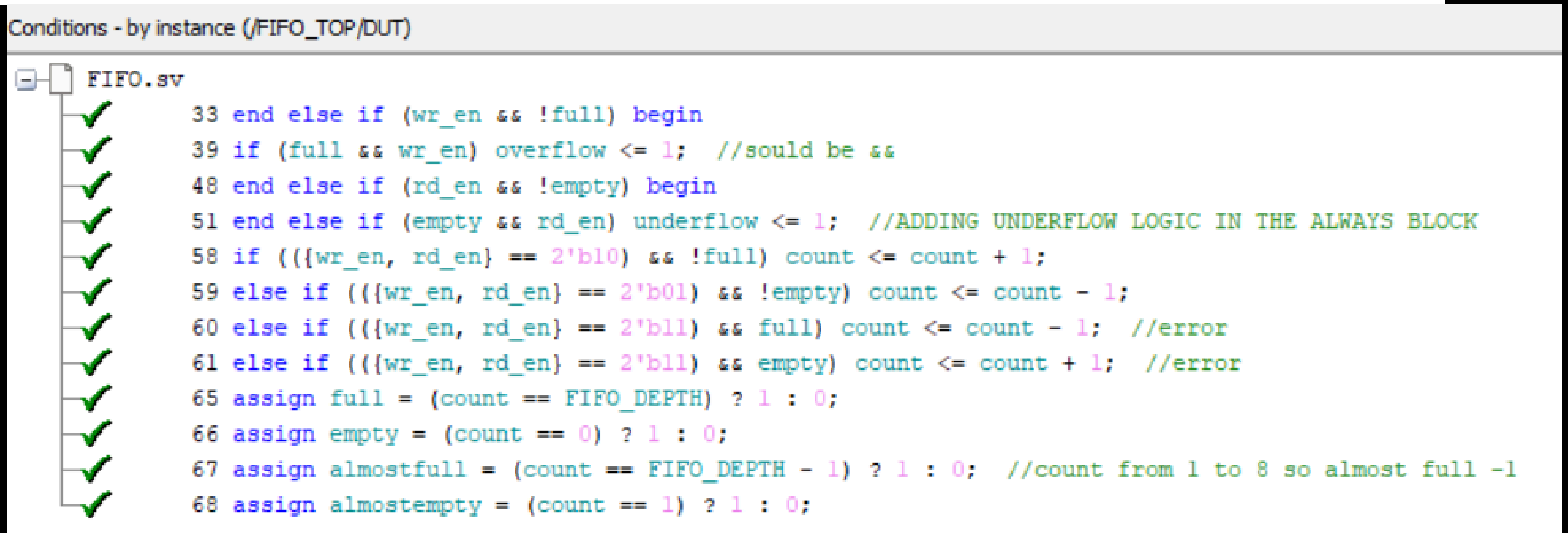
```
always @(posedge clk or negedge rst_n) begin
wr_ptr   <= 0;
overflow <= 0;  //since overflow is a seq so reset affects it
wr_ack   <= 0;  //wr_ack should equal zero in reset
mem[wr_ptr] <= data_in;
wr_ack <= 1;
wr_ptr <= wr_ptr + 1;
wr_ack <= 0;
if (full && wr_en) overflow <= 1;  //sould be &&
else overflow <= 0;
always @(posedge clk or negedge rst_n) begin
rd_ptr <= 0;
underflow <= 0;  //reseting underflow since it is a seq logic
data_out <= mem[rd_ptr];
rd_ptr   <= rd_ptr + 1;
end else if (empty && rd_en) underflow <= 1;  //ADDING UNDERFLOW LOGI
else underflow <= 0;  //ASSERT TO 0 IF NO EMPTY AND NO read_enable
always @(posedge clk or negedge rst_n) begin
count  <= 0;
if (({wr_en, rd_en} == 2'b10) && !full) count <= count + 1;
else if (({wr_en, rd_en} == 2'b01) && !empty) count <= count - 1;
else if (({wr_en, rd_en} == 2'b11) && full) count <= count - 1;  //error
else if (({wr_en, rd_en} == 2'b11) && empty) count <= count + 1;  //error
assign full = (count == FIFO_DEPTH) ? 1 : 0;
assign empty = (count == 0) ? 1 : 0;
assign almostfull = (count == FIFO_DEPTH - 1) ? 1 : 0;  //count from 1 to 8 so almost full -1
assign almostempty = (count == 1) ? 1 : 0;
```



Branches - by instance (/FIFO_TOP/DUT)

FIFO.sv

```
if (!rst_n) begin
end else if (wr_en && !full) begin
end else begin
if (full && wr_en) overflow <= 1;  //sould be &&
else overflow <= 0;
if (!rst_n) begin
end else if (rd_en && !empty) begin
end else if (empty && rd_en) underflow <= 1;  //ADDING UNDERFLOW LOGIC IN THE ALWAYS BLOCK
else underflow <= 0;  //ASSERT TO 0 IF NO EMPTY AND NO read_enable
if (!rst_n) begin
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;
else if (({wr_en, rd_en} == 2'b11) && full) count <= count - 1;  //error
else if (({wr_en, rd_en} == 2'b11) && empty) count <= count + 1;  //error
assign full = (count == FIFO_DEPTH) ? 1 : 0;
assign empty = (count == 0) ? 1 : 0;
assign almostfull = (count == FIFO_DEPTH - 1) ? 1 : 0;  //count from 1 to 8 so almost full -1
assign almostempty = (count == 1) ? 1 : 0;
```



Conditions - by instance (/FIFO_TOP/DUT)

FIFO.sv

```
end else if (wr_en && !full) begin
if (full && wr_en) overflow <= 1;  //sould be &&
end else if (rd_en && !empty) begin
end else if (empty && rd_en) underflow <= 1;  //ADDING UNDERFLOW LOGIC IN THE ALWAYS BLOCK
if (({wr_en, rd_en} == 2'b10) && !full) count <= count + 1;
else if (({wr_en, rd_en} == 2'b01) && !empty) count <= count - 1;
else if (({wr_en, rd_en} == 2'b11) && full) count <= count - 1;  //error
else if (({wr_en, rd_en} == 2'b11) && empty) count <= count + 1;  //error
assign full = (count == FIFO_DEPTH) ? 1 : 0;
assign empty = (count == 0) ? 1 : 0;
assign almostfull = (count == FIFO_DEPTH - 1) ? 1 : 0;  //count from 1 to 8 so almost full -1
assign almostempty = (count == 1) ? 1 : 0;
```

COVERAGE REPORT ON GITHUB