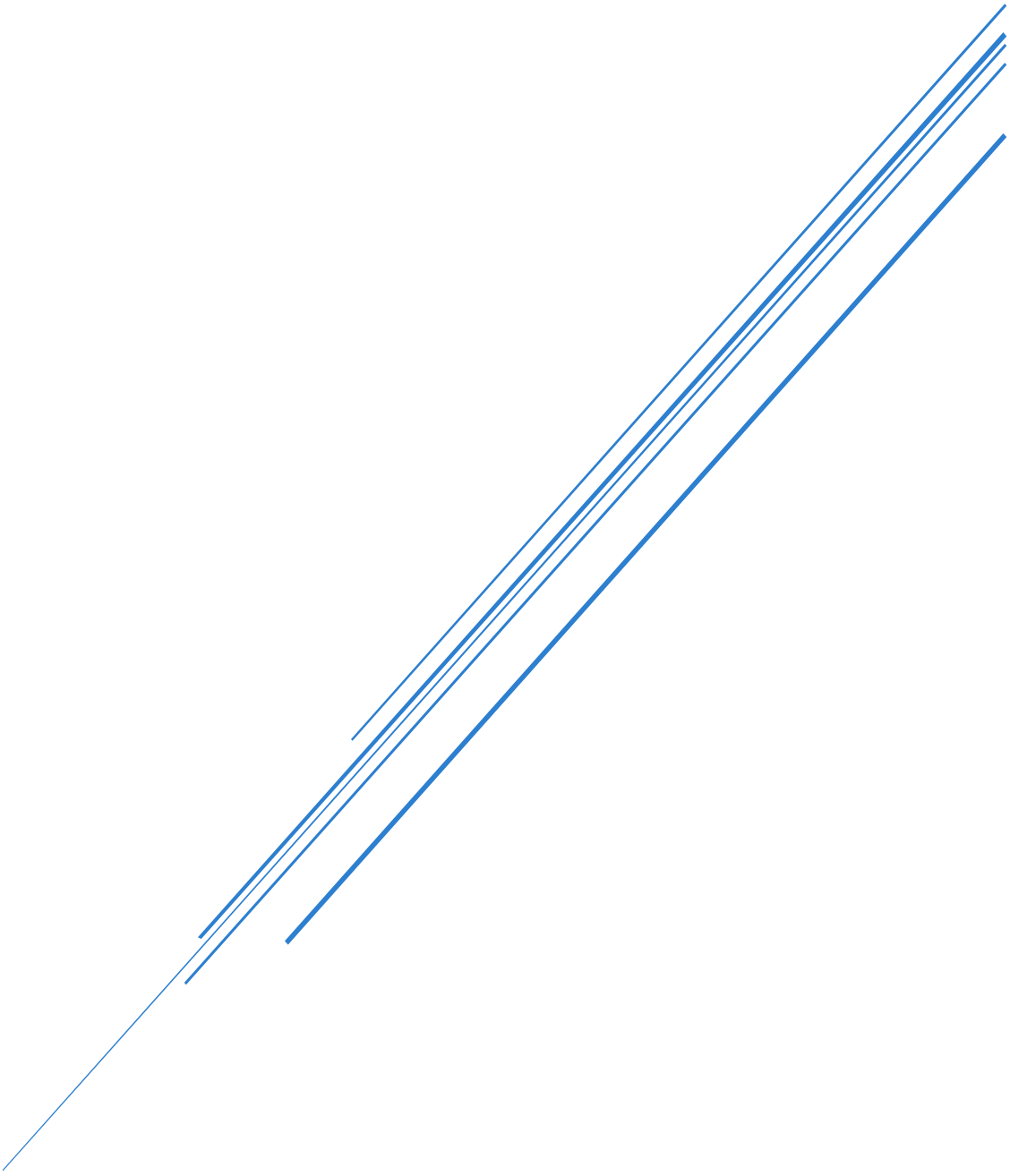
System Verilog project

FIFO

Digital Verification



Name: Nabil Ebrahim Abd_Elaty Abd_Elrazeq

TA: Mai Sherif

Table of contents

First: Snippets from codes:

The Interface:

```
interface fifo_if (clk);

parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;
input bit clk;
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, underflow;
logic full, empty, almostfull, almostempty;

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

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

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

endinterface
```

Top module:

```
module fifo_top;

bit clk;

initial begin
   clk = 0;
   forever #1 clk = ~clk;
end

fifo_if f_if (clk);
fifo_tb       f_tb (f_if.TEST);
fifo          f_dut(f_if.DUT);
fifo_monitor f_mon(f_if.MONITOR);

endmodule
```

Shared package:

```
package shared_pkg;

    bit     test_finished = 0;
    int     error_count = 0;
    int     correct_count = 0;
    event   sample_event;

endpackage
```

Do and source files:

```
vlib work

vlog -f src_files.list +cover +covercells +coveropt

vsim -voptargs=+acc work.fifo_top -cover

add wave *

coverage save fifo_top.ucdb -onexit

run -all

## vcover report fifo_top.ucdb -details -annotate -all -output coverage_pro_rpt.txt
```

```
FIFO_IF.sv
FIFO_TOP.sv
FIFO_SHARED.sv
FIFO.sv
FIFO_PKG_TRANS.sv
FIFO_PKG_COV.sv
FIFO_PKG_SCORE.sv
FIFO_MONITOR.sv
FIFO_TB.sv
```

Design and assertions module:

```
module fifo (fifo_if.DUT f_if);

    // ------------------------------------------------
    // Local parameters (taken from interface)
    // ------------------------------------------------
    localparam FIFO_WIDTH    = f_if.FIFO_WIDTH;
    localparam FIFO_DEPTH    = f_if.FIFO_DEPTH;
    localparam max_fifo_addr = $clog2(FIFO_DEPTH);

    // ------------------------------------------------
    // Internal registers
    // ------------------------------------------------
    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;

    // ------------------------------------------------
    // WRITE logic
    // ------------------------------------------------
    always @(posedge f_if.clk or negedge f_if.rst_n) begin
        if (!f_if.rst_n) begin
            wr_ptr         <= 0;
            // Bug detected: Reset signals FIFO_IF.overflow & FIFO_IF.wr_ack
            f_if.wr_ack    <= 0;
            f_if.overflow  <= 0;
        end
        else if (f_if.wr_en && !f_if.full) begin
            mem[wr_ptr]    <= f_if.data_in;
            wr_ptr         <= wr_ptr + 1;
            f_if.wr_ack    <= 1;
            f_if.overflow  <= 0;
        end
        else begin
            f_if.wr_ack    <= 0;
            if (f_if.wr_en && f_if.full)
                f_if.overflow <= 1;   // reject write when full
            else
                f_if.overflow <= 0;
        end
    end

    // ------------------------------------------------
    // READ logic
    // ------------------------------------------------
    always @(posedge f_if.clk or negedge f_if.rst_n) begin
        if (!f_if.rst_n) begin
            rd_ptr          <= 0;
            // Bug detected: Reset signals FIFO_IF.underflow
            f_if.data_out   <= 0;
            f_if.underflow  <= 0;
        end
        else if (f_if.rd_en && !f_if.empty) begin
            f_if.data_out   <= mem[rd_ptr];
            rd_ptr          <= rd_ptr + 1;
            f_if.underflow  <= 0;
        end
        else begin
            if (f_if.rd_en && f_if.empty)
                f_if.underflow <= 1;   // reject read when empty
            else
                f_if.underflow <= 0;
        end
    end

    // ------------------------------------------------
    // COUNT logic
    // ------------------------------------------------
    always @(posedge f_if.clk or negedge f_if.rst_n) begin
        if (!f_if.rst_n) begin
            count <= 0;
        end
        else begin
            case ({f_if.wr_en, f_if.rd_en})
                2'b10: if (!f_if.full)  count <= count + 1; // write only
                2'b01: if (!f_if.empty) count <= count - 1; // read only
                2'b11: begin
    // simultaneous read and write
// Bug detected: Unhandled case,  If a read and write enables were high and the FIFO was FIFO_IF.empty, only writing will take place.
// Bug detected: Unhandled cases,  If a read and write enables were high and the FIFO was FIFO_IF.full, only reading will take place.
                    if (f_if.empty)      count <= count + 1; // write takes effect
                    else if (f_if.full)  count <= count - 1; // read takes effect
                    // else: count unchanged (balanced read/write)
                end
                default: count <= count; // no operation
            endcase
        end
    end
```

```
    // ------------------------------------------------
    // Status signals
    // ------------------------------------------------
    assign f_if.full        = (count == FIFO_DEPTH);
    assign f_if.empty       = (count == 0);
    assign f_if.almostfull  = (count == FIFO_DEPTH-1); // Bug detected: f_if.FIFO_DEPTH-2 --> f_if.FIFO_DEPTH-1

    assign f_if.almostempty = (count == 1);

  always_comb begin
    if(!f_if.rst_n) begin
    reset_rd :assert final(rd_ptr==1'b0);
    reset_wr :assert final(wr_ptr==1'b0);
    reset_co :assert final(count==0);
    end
end

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

property ovr;
    @(posedge f_if.clk) disable iff (!f_if.rst_n) (f_if.wr_en && f_if.full) |=> f_if.overflow;
endproperty

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

property emp;
    @(posedge f_if.clk) disable iff (!f_if.rst_n) count==0 |-> f_if.empty;
endproperty

property ful;
    @(posedge f_if.clk) disable iff (!f_if.rst_n) count==FIFO_DEPTH |-> f_if.full;
endproperty

property Aful;
    @(posedge f_if.clk) disable iff (!f_if.rst_n) (count==FIFO_DEPTH-1) |-> f_if.almostfull;
endproperty

property Aemp;
    @(posedge f_if.clk) disable iff (!f_if.rst_n) (count==1) |-> f_if.almostempty;
endproperty

property wrap1;
     @(posedge f_if.clk) disable iff (!f_if.rst_n) (f_if.wr_en && !f_if.full && wr_ptr == FIFO_DEPTH-1) |=> (wr_ptr == 0);
endproperty

property wrap2;
     @(posedge f_if.clk) disable iff (!f_if.rst_n) (f_if.rd_en && !f_if.empty && rd_ptr == FIFO_DEPTH-1) |=> (rd_ptr == 0);
endproperty

property wrap3;
     @(posedge f_if.clk) disable iff (!f_if.rst_n) (count<=FIFO_DEPTH && count>=0);
endproperty
property FIFO_internal_bounds;
  @(posedge f_if.clk) disable iff (!f_if.rst_n) (wr_ptr < FIFO_DEPTH) && (rd_ptr < FIFO_DEPTH) && (count <= FIFO_DEPTH);
endproperty

assert property (ack);
assert property (ovr);
assert property (udr);
assert property (emp);
assert property (ful);
assert property (Aful);
assert property (Aemp);
assert property (wrap1);
assert property (wrap2);
assert property (wrap3);
assert property (FIFO_internal_bounds);
cover property (ack);
cover property (ovr);
cover property (udr);
cover property (emp);
cover property (ful);
cover property (Aful);
cover property (Aemp);
cover property (wrap1);
cover property (wrap2);
cover property (wrap3);
cover property (FIFO_internal_bounds);

endmodule
```

Transaction package:

```
package fifo_pkg_trans;
import shared_pkg::*;

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

    rand logic [15:0] data_in;
    rand logic wr_en;
    rand logic rd_en;
    rand logic rst_n;

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

    int RD_EN_ON_DIST;
    int WR_EN_ON_DIST;
    function new(int rd_on = 30, int wr_on = 70);
      this.RD_EN_ON_DIST = rd_on;
      this.WR_EN_ON_DIST = wr_on;
    endfunction

    constraint rst_less_often {
      rst_n dist {1 :/ 95, 0 :/ 5};
    }

    constraint wr_dist {
      wr_en dist {1 :/ WR_EN_ON_DIST, 0 :/ (100-WR_EN_ON_DIST)};
    }
    constraint rd_dist {
      rd_en dist {1 :/ RD_EN_ON_DIST, 0 :/ (100-RD_EN_ON_DIST)};
    }

endclass

endpackage
```

Coverage package:

```
package fifo_pkg_COV;
import fifo_pkg_trans::*;

class FIFO_coverage;
FIFO_transaction F_cvg_txn;

    covergroup cg;

      cp_wr_en : coverpoint F_cvg_txn.wr_en;
      cp_rd_en : coverpoint F_cvg_txn.rd_en;

      cp_wr_ack: coverpoint F_cvg_txn.wr_ack;
      cp_overflow: coverpoint F_cvg_txn.overflow;
      cp_underflow: coverpoint F_cvg_txn.underflow;
      cp_full: coverpoint F_cvg_txn.full;
      cp_empty: coverpoint F_cvg_txn.empty;
      cp_almostfull: coverpoint F_cvg_txn.almostfull;
      cp_almostempty: coverpoint F_cvg_txn.almostempty;

      cross_wre_rd_wrack: cross cp_wr_en, cp_rd_en, cp_wr_ack {
          // wr_ack can only be 1 when wr_en is 1
          illegal_bins wr_ack_without_wr_en = binsof(cp_wr_en) intersect {0} &&
                                              binsof(cp_wr_ack) intersect {1};
      }

      cross_wre_rd_oflow: cross cp_wr_en, cp_rd_en, cp_overflow {
          // overflow can only be 1 when wr_en is 1
          illegal_bins overflow_without_wr_en = binsof(cp_wr_en) intersect {0} &&
                                                binsof(cp_overflow) intersect {1};
      }

      cross_wre_rd_uflow: cross cp_wr_en, cp_rd_en, cp_underflow {
          // underflow can only be 1 when rd_en is 1
          illegal_bins underflow_without_rd_en = binsof(cp_rd_en) intersect {0} &&
                                                 binsof(cp_underflow) intersect {1};
      }

      cross_wre_rd_full: cross cp_wr_en, cp_rd_en, cp_full {
          illegal_bins full_with_only_read = binsof(cp_rd_en) intersect {1} &&
                                             binsof(cp_full) intersect {1};
      }

      cross_wre_rd_empty: cross cp_wr_en, cp_rd_en, cp_empty;

      cross_wre_rd_afull: cross cp_wr_en, cp_rd_en, cp_almostfull;

      cross_wre_rd_aempty: cross cp_wr_en, cp_rd_en, cp_almostempty;

    endgroup : cg


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

   function void sample_data(FIFO_transaction F_txn);
      this.F_cvg_txn = F_txn;

      cg.sample();
    endfunction

endclass


endpackage
```

Scoreboard package:

```
package fifo_pkg_score;
import fifo_pkg_trans::*;
import shared_pkg::*;

FIFO_transaction FIFO_transaction_object = new();

class FIFO_scoreboard;

    bit [FIFO_transaction_object.FIFO_WIDTH-1:0] fifo_q[$];

    logic [FIFO_transaction_object.FIFO_WIDTH-1:0] data_out_ref;
    logic full_ref, empty_ref, almostfull_ref, almostempty_ref;
    logic wr_ack_ref, overflow_ref, underflow_ref;
    int count_ref;

function void reference_model(FIFO_transaction tr);

  if (!tr.rst_n) begin
    fifo_q.delete();
    data_out_ref    = '0;
    full_ref        = 0;
    empty_ref       = 1;
    almostfull_ref  = 0;
    almostempty_ref = 0;
    count_ref       = 0;
    return;
  end  else begin
  if (tr.wr_en && (!full_ref)) begin
    fifo_q.push_back(tr.data_in);
  end
  if ( tr.rd_en && (!empty_ref)) begin
    data_out_ref = fifo_q.pop_front();
  end

        count_ref       = fifo_q.size();
        full_ref        = (count_ref == tr.FIFO_DEPTH);
        empty_ref       = (count_ref == 0);
 end
endfunction


    function void check_data(input FIFO_transaction ob1);
        reference_model(ob1);

      if ((ob1.data_out !== data_out_ref)&&(ob1.wr_en)&&(ob1.rst_n)) begin
        $display("Error!!, At time %t, data_out %d doesn't equal data_out_ref %d !!", $time, ob1.data_out, data_out_ref);
        error_count++;
      end
      else begin
        $display("Success, At time %t, data_out= %d equals data_out_ref= %d", $time, ob1.data_out, data_out_ref);
        correct_count++;
      end
    endfunction

        function new();
    data_out_ref    = '0;
    full_ref        = 0;
    empty_ref       = 1;
    almostfull_ref  = 0;
    almostempty_ref = 0;
    count_ref       = 0;
        endfunction

  endclass
endpackage
```

Monitor module:

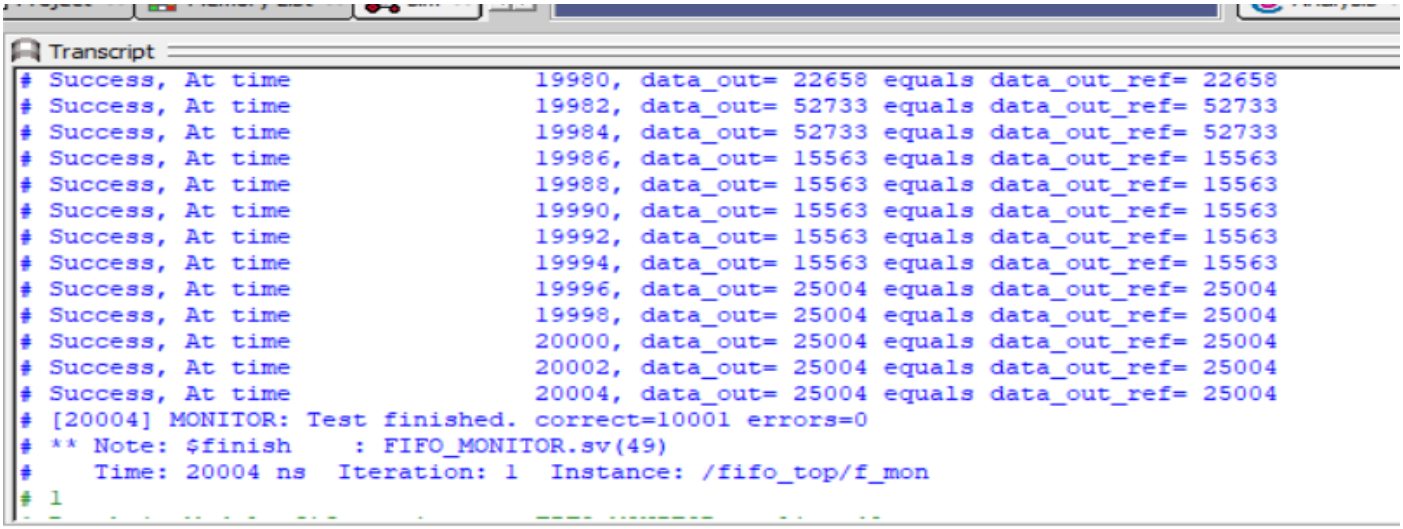
```
  import fifo_pkg_trans::*;
  import fifo_pkg_COV::*;
  import fifo_pkg_score::*;
  import shared_pkg::*;

module fifo_monitor(fifo_if.MONITOR f_if);

  FIFO_transaction mon_txn;
  FIFO_coverage  mon_cvg;
  FIFO_scoreboard mon_sb;

   initial begin

    mon_txn = new();
    mon_cvg = new();
    mon_sb  = new();

    forever begin
     @(sample_event);
      mon_txn.data_in    = f_if.data_in;
      mon_txn.wr_en      = f_if.wr_en;
      mon_txn.rd_en      = f_if.rd_en;
      mon_txn.rst_n      = f_if.rst_n;

      @(negedge f_if.clk);

      mon_txn.data_out   = f_if.data_out;
      mon_txn.wr_ack     = f_if.wr_ack;
      mon_txn.overflow   = f_if.overflow;
      mon_txn.underflow  = f_if.underflow;
      mon_txn.full       = f_if.full;
      mon_txn.empty      = f_if.empty;
      mon_txn.almostfull = f_if.almostfull;
      mon_txn.almostempty= f_if.almostempty;


      fork
        begin
          mon_cvg.sample_data(mon_txn);
        end
        begin
          mon_sb.check_data(mon_txn);
        end
      join


      if (test_finished) begin
        $display("[%0t] MONITOR: Test finished. correct=
 errors=$finish;
", $time, correct_count, error_count);
    end
   end

endmodule
```

Testbench module:

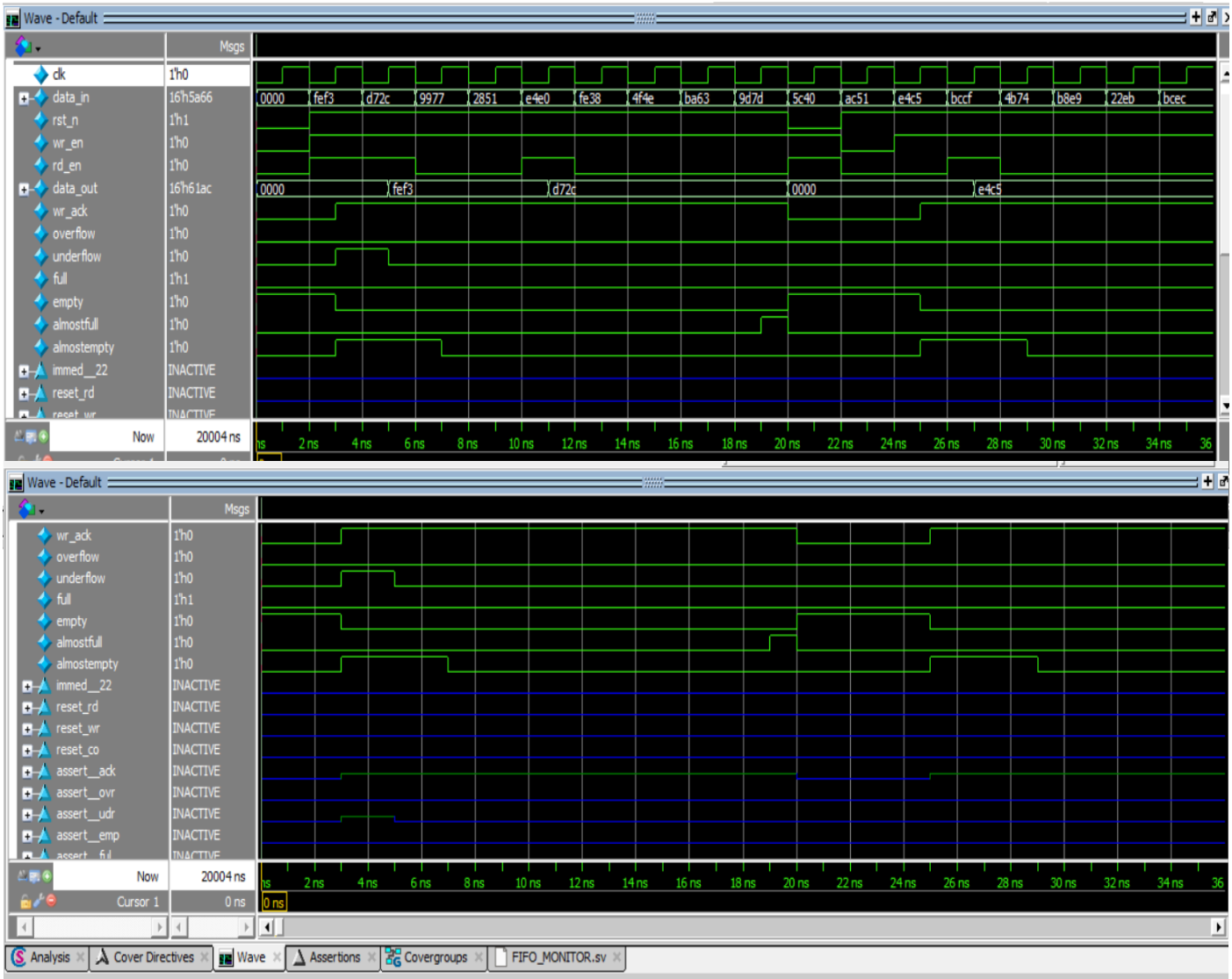
```
module fifo_tb(fifo_if.TEST f_if);

  import fifo_pkg_trans::*;
  import shared_pkg::*;
  import fifo_pkg_COV::*;
  import fifo_pkg_score::*;

  FIFO_transaction drv_txn;

  int unsigned NUM_CYCLES = 10000;

  initial begin

    drv_txn = new();

    f_if.rst_n=0; f_if.rd_en=0; f_if.wr_en=0; f_if.data_in=0;
    @(negedge f_if.clk);
    -> sample_event;
    f_if.rst_n=1;

    repeat(NUM_CYCLES) begin
        assert(drv_txn.randomize());
        f_if.rst_n=drv_txn.rst_n;
        f_if.rd_en=drv_txn.rd_en;
        f_if.wr_en=drv_txn.wr_en;
        f_if.data_in=drv_txn.data_in;
        @(negedge f_if.clk);
        -> sample_event;
    end

    test_finished =1;

  end

endmodule
```

Second: Snippets from Simulation results:

Transcript:

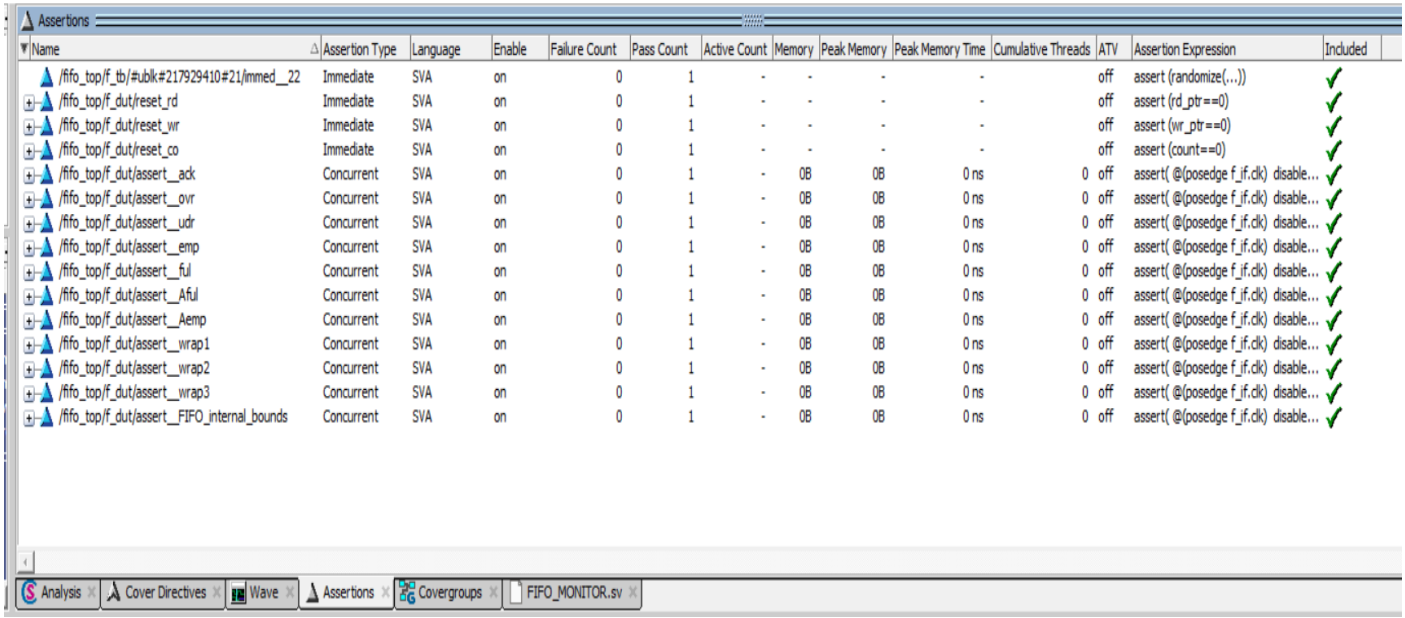


Waveforms:

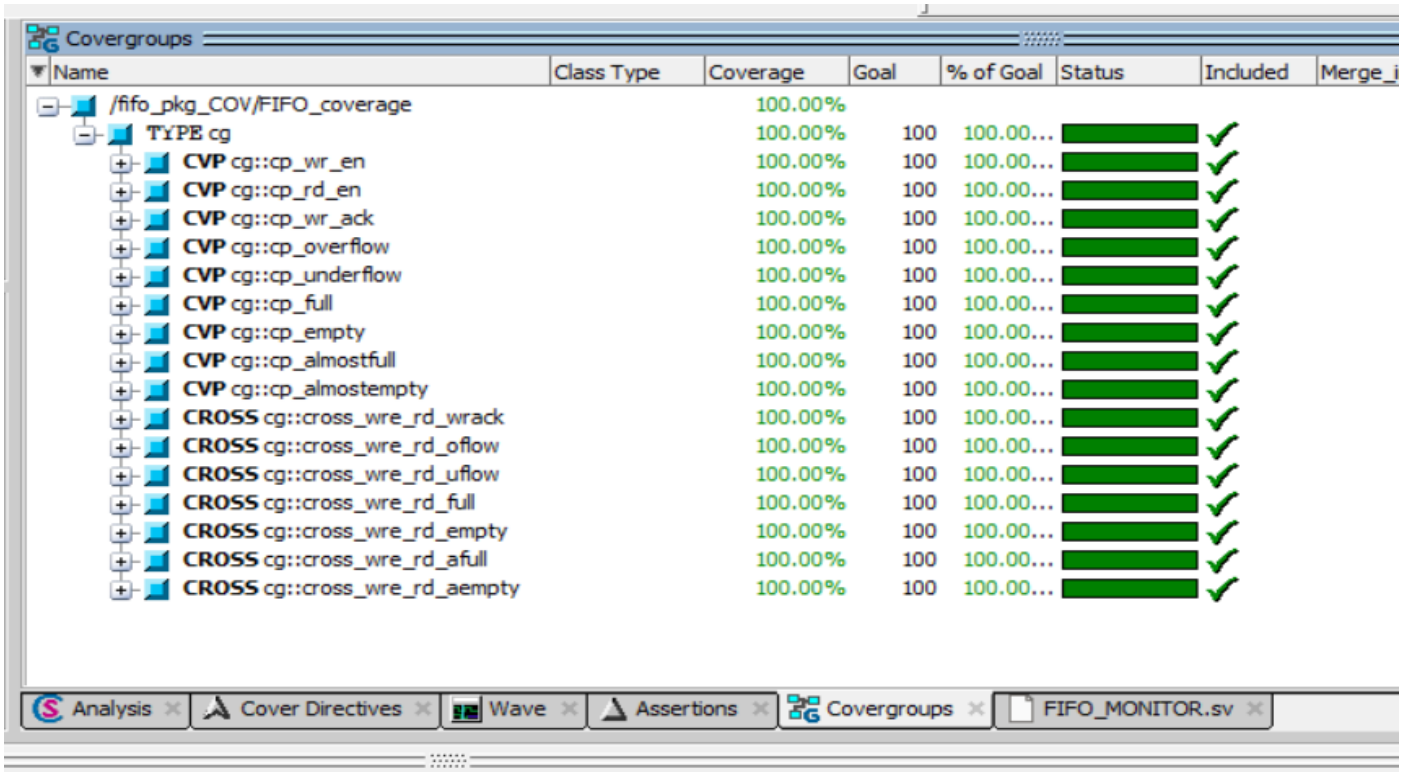


Assertions coverage:

```
================================================================================
=== Instance: /fifo_top/f_dut
=== Design Unit: work.fifo
================================================================================

Assertion Coverage:
    Assertions                      14        14         0   100.00%
--------------------------------------------------------------------
```



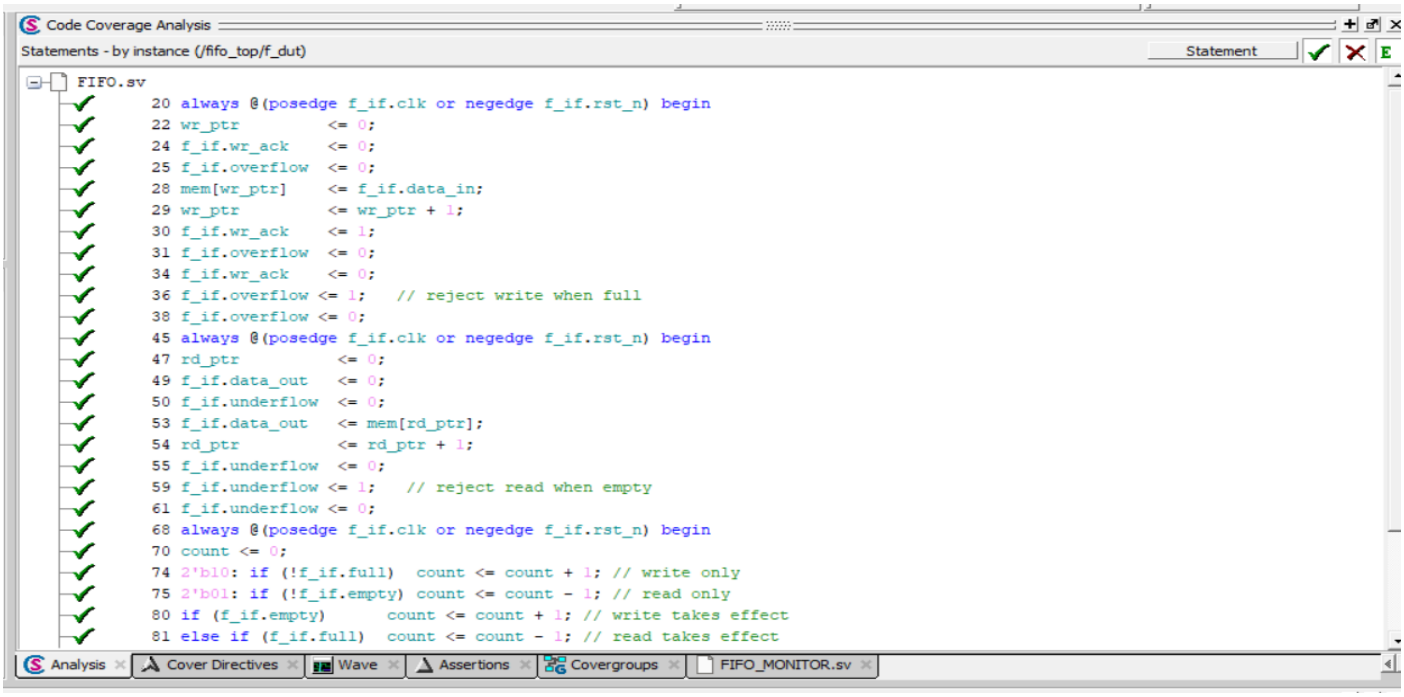
Function coverage:

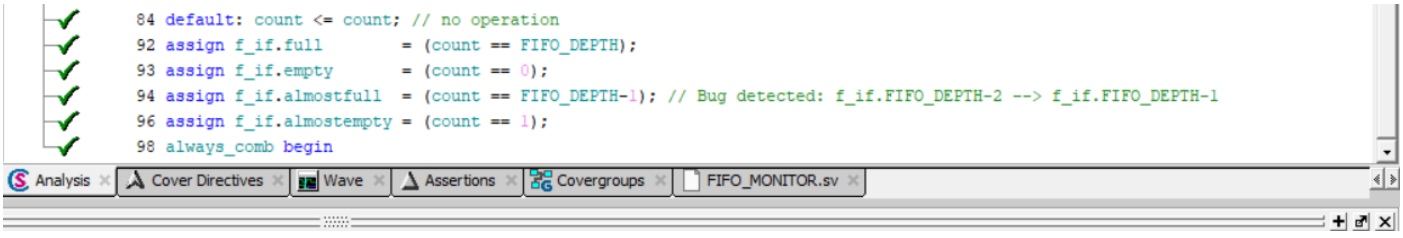


```
================================================================================
=== Instance: /fifo_pkg_COV
=== Design Unit: work.fifo_pkg_COV
================================================================================

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

Code coverage:

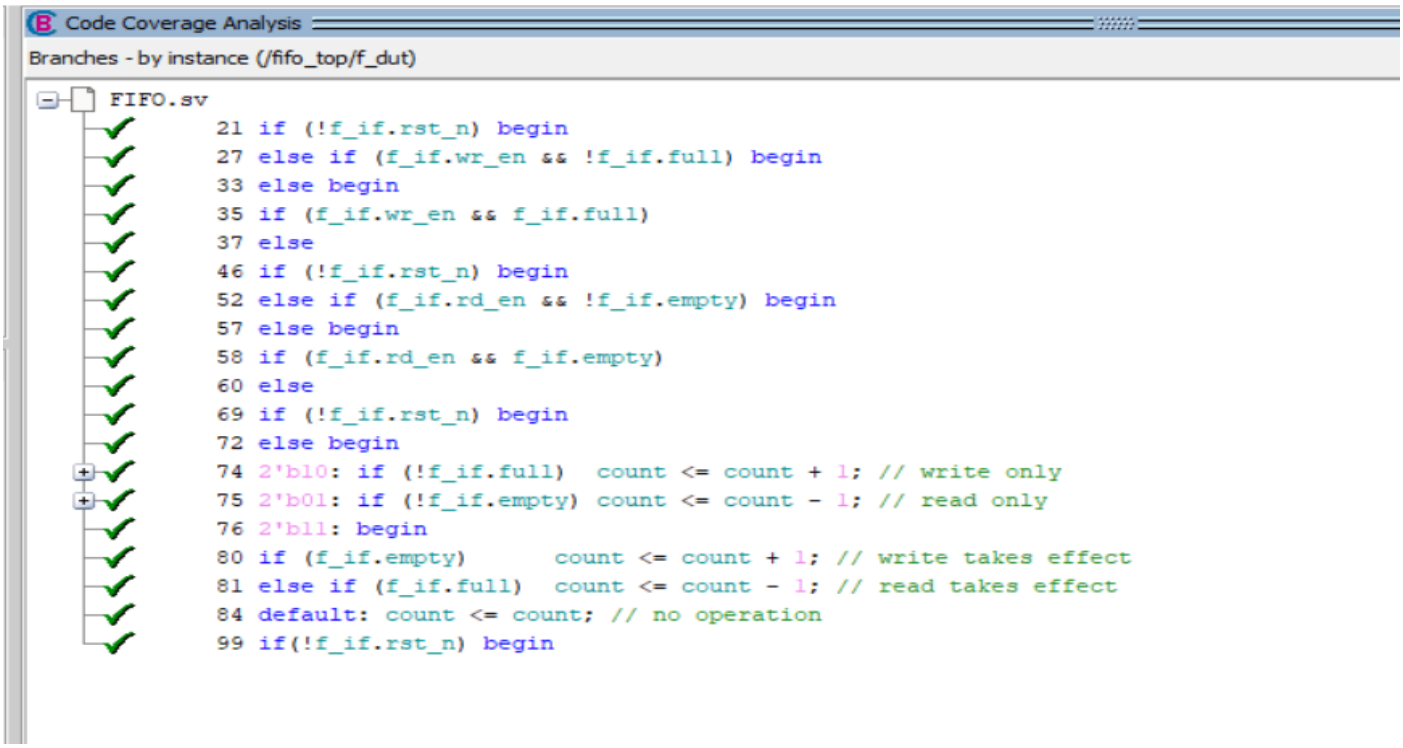
-Statements:





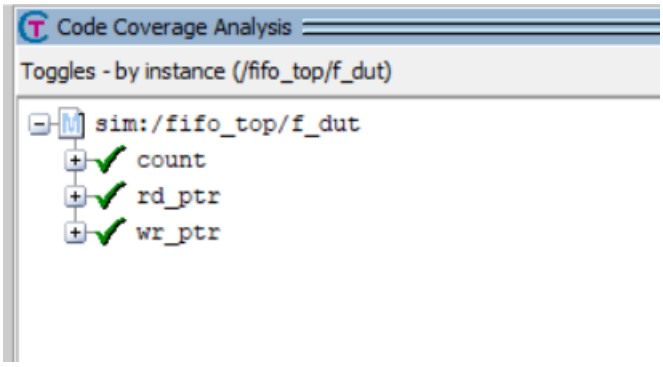

```
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                      32        32         0   100.00%

================================Statement Details================================
```

-Branches:




```
FIFO.sv(102)                                0          1
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        25        25         0   100.00%

================================Branch Details================================
```

-Toggles:




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

================================Toggle Details================================
```

Third: Snippet from The Verification plan:

Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
RESET_1	When reset is asserted (rst_n=0), all internal registers (wr_ptr, rd_ptr, count) should be cleared to 0	Apply rst_n=0 at start of simulation, then randomize with 5% probability during test using constraint rst_less_often	Cover reset assertion with all possible FIFO states (empty, full, partial)	Assert that wr_ptr=0, rd_ptr=0, count=0, empty=1, full=0 immediately after reset
RESET_2	When reset is asserted, all output signals (wr_ack, overflow, underflow, data_out) should be cleared	Apply rst_n=0 with various combinations of wr_en and rd_en active	Cover reset with active write/read attempts	Assert that wr_ack=0, overflow=0, underflow=0, data_out=0 during reset
WRITE_1	When wr_en=1 and FIFO is not full, data should be written to memory at wr_ptr location	Randomize data_in with wr_en=1 using 70% distribution, ensure FIFO not full	Cover write operations at different fill levels: empty, partial, almost full	Check that wr_ack=1 on next cycle and data is stored correctly in reference model
WRITE_2	When wr_en=1 and FIFO is not full, wr_ptr should increment (with wrap-around at FIFO_DEPTH-1)	Generate consecutive writes until FIFO is full	Cover wr_ptr increment from 0 to FIFO_DEPTH-1, cover wrap-around case	Assert property wrap1: wr_ptr wraps to 0 after reaching FIFO_DEPTH-1
WRITE_3	When wr_en=1 and FIFO is not full, count should increment by 1	Generate single writes with no reads	Cover count transitions from 0 to FIFO_DEPTH	Verify count increments correctly in scoreboard reference model
WRITE_4	When wr_en=1 and FIFO is full, overflow flag should be set and write rejected	Generate write attempts when count=FIFO_DEPTH	Cover overflow condition: cross cp_wr_en, cp_full, cp_overflow	Assert property ovr: overflow=1 when wr_en=1 and full=1
READ_1	When rd_en=1 and FIFO is not empty, data should be read from memory at rd_ptr location	Randomize rd_en=1 using 30% distribution, ensure FIFO not empty	Cover read operations at different fill levels: full, partial, almost empty	Check that data_out matches expected value from reference queue
READ_2	When rd_en=1 and FIFO is not empty, rd_ptr should increment (with wrap-around at FIFO_DEPTH-1)	Generate consecutive reads until FIFO is empty	Cover rd_ptr increment from 0 to FIFO_DEPTH-1, cover wrap-around case	Assert property wrap2: rd_ptr wraps to 0 after reaching FIFO_DEPTH-1
READ_3	When rd_en=1 and FIFO is not empty, count should decrement by 1	Generate single reads with no writes	Cover count transitions from FIFO_DEPTH down to 0	Verify count decrements correctly in scoreboard reference model
READ_4	When rd_en=1 and FIFO is empty, underflow flag should be set and read rejected	Generate read attempts when count=0	Cover underflow condition: cross cp_rd_en, cp_empty, cp_underflow	Assert property udr: underflow=1 when rd_en=1 and empty=1
SIMUL_WR_RD_1	When wr_en=1 and rd_en=1 simultaneously and FIFO is neither empty nor full, count should remain unchanged	Generate simultaneous wr_en=1 and rd_en=1 when 0<count<FIFO_DEPTH	Cross coverage: cp_wr_en=1, cp_rd_en=1 with various count values	Verify count remains stable and both operations succeed
SIMUL_WR_RD_2	When wr_en=1 and rd_en=1 simultaneously and FIFO is empty, only write should occur and count should increment	Generate simultaneous wr_en=1 and rd_en=1 when count=0	Cross coverage: cp_wr_en=1, cp_rd_en=1, cp_empty=1	Verify count increments by 1, underflow=0, write succeeds
SIMUL_WR_RD_3	When wr_en=1 and rd_en=1 simultaneously and FIFO is full, only read should occur and count should decrement	Generate simultaneous wr_en=1 and rd_en=1 when count=FIFO_DEPTH	Cross coverage: cp_wr_en=1, cp_rd_en=1, cp_full=1	Verify count decrements by 1, overflow=0, read succeeds
STATUS_FULL	Full flag should be asserted when count equals FIFO_DEPTH	Fill FIFO completely by writing FIFO_DEPTH times without reads	Cover full flag assertion: cp_full=1	Assert property ful: full=1 when count=FIFO_DEPTH
STATUS_EMPTY	Empty flag should be asserted when count equals 0	Empty FIFO completely by reading all entries	Cover empty flag assertion: cp_empty=1	Assert property emp: empty=1 when count=0

STATUS_AFULL	Almost full flag should be asserted when count equals FIFO_DEPTH-1	Write FIFO_DEPTH-1 entries	Cover almost full: cross_wre_rd_afull with count=FIFO_DEPTH-1	Assert property Aful: almostfull=1 when count=FIFO_DEPTH-1
STATUS_AEMPTY	Almost empty flag should be asserted when count equals 1	Read until only 1 entry remains	Cover almost empty: cross_wre_rd_aempty with count=1	Assert property Aemp: almostempty=1 when count=1
WR_ACK	Write acknowledge should be asserted only when wr_en=1 and FIFO is not full	Generate various wr_en patterns with different FIFO states	Cover wr_ack with illegal bins: wr_ack=1 when wr_en=0	Assert property ack: wr_ack=1 follows successful write
DATA_INTEGRITY	Data written to FIFO should match data read out in FIFO order	Write sequence of unique random data, then read back	Cover full range of data_in values (16-bit)	Scoreboard compares data_out with expected values from reference queue
PTR_BOUNDS	Write and read pointers should always remain within valid range [0, FIFO_DEPTH-1]	Generate extensive random sequences over 10000 cycles	Cover boundary cases: wr_ptr and rd_ptr at min/max values	Assert property FIFO_internal_bounds: wr_ptr<FIFO_DEPTH and rd_ptr<FIFO_DEPTH
COUNT_BOUNDS	Count should always remain within valid range [0, FIFO_DEPTH]	Generate random write/read patterns	Cover all count values from 0 to FIFO_DEPTH	Assert property wrap3: 0<=count<=FIFO_DEPTH at all times
STRESS_TEST	FIFO should handle continuous random operations without errors	Run 10000 random cycles with 70% write / 30% read distribution	Achieve 100% coverage on all coverpoints and crosses	Monitor reports correct_count vs error_count, expect 0 errors

الحمد لله