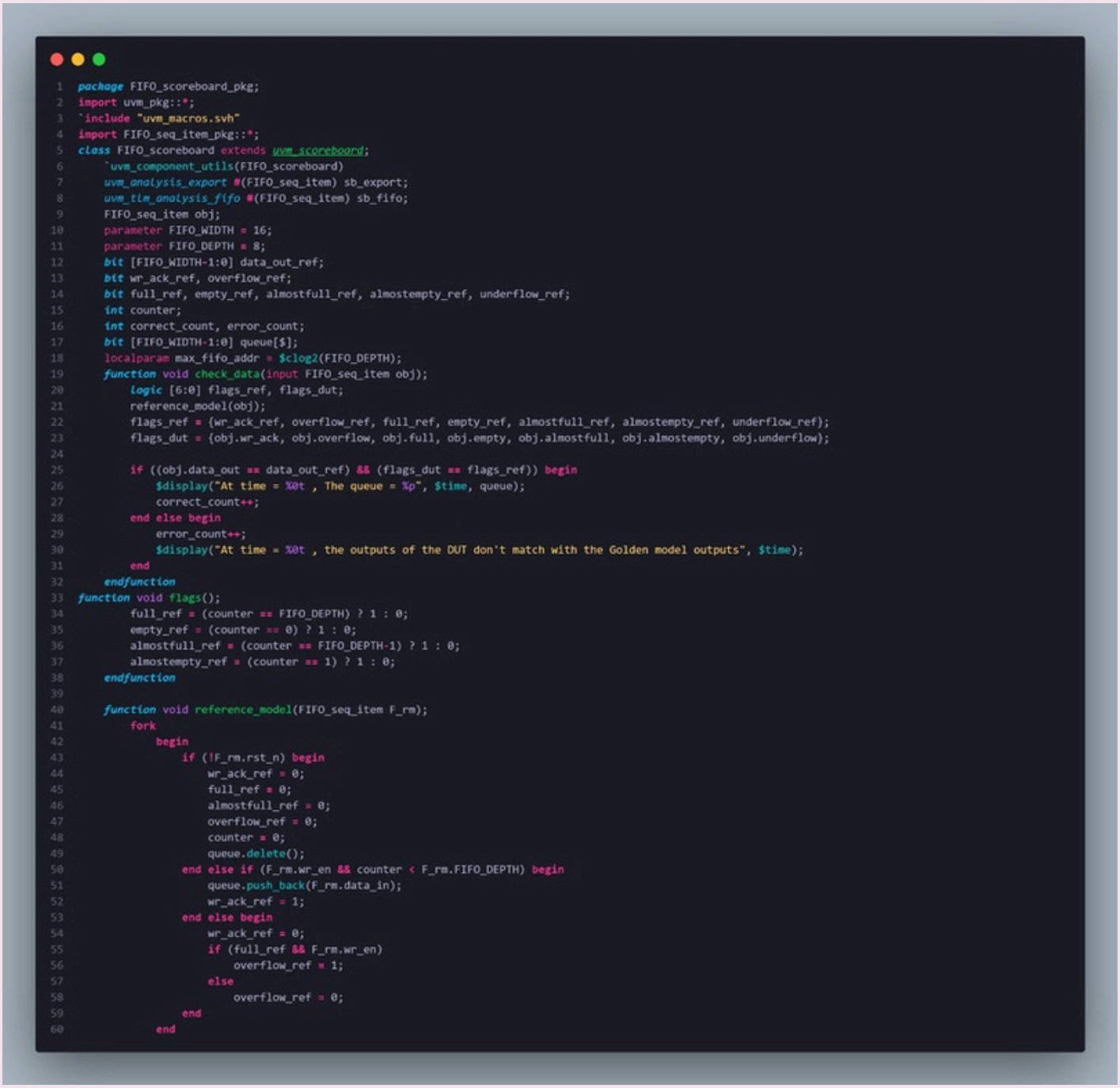
MARIAM MOHAMED MARZOUK

PROJECT

FIFO:

```
module FIFO(FIFO_if.DUT fif);
localparam max_fifo_addr = $clog2(fif.FIFO_DEPTH);
reg [fif.FIFO_WIDTH-1:0] mem [fif.FIFO_DEPTH-1:0];
reg [max_fifo_addr-1:0] wr_ptr, rd_ptr;
reg [max_fifo_addr:0] count;
always @(posedge fif.clk or negedge fif.rst_n) begin
    if (!fif.rst_n) begin
        wr_ptr <= 0;
        fif.overflow <= 0;
        fif.wr_ack <= 0;
    end
    else if (fif.wr_en && count < fif.FIFO_DEPTH) begin
        mem[wr_ptr] <= fif.data_in;
        fif.wr_ack <= 1;
        wr_ptr <= wr_ptr + 1;
    end else begin
        fif.wr_ack <= 0;
        if (fif.full && fif.wr_en)
            fif.overflow <= 1;
        else fif.overflow <= 0;
    end end
always @(posedge fif.clk or negedge fif.rst_n) begin
    if (!fif.rst_n) begin
        rd_ptr <= 0;
        fif.underflow <= 0;
        fif.data_out <= 0;
    end else if (fif.rd_en && count != 0) begin
        fif.data_out <= mem[rd_ptr];
        rd_ptr <= rd_ptr + 1;
        end
    else begin
        if(fif.empty && fif.rd_en)
            fif.underflow  <= 1;
        else
            fif.underflow  <= 0;
    end end
// always block specialized for counter signal
always @(posedge fif.clk or negedge fif.rst_n) begin
    if (!fif.rst_n) begin
        count <= 0;
    end
    else begin
        if  ( ({fif.wr_en, fif.rd_en} == 2'b10) && !fif.full)
            count <= count + 1;
        else if ( ({fif.wr_en, fif.rd_en} == 2'b01) && !fif.empty)
            count <= count - 1;
        else if (({fif.wr_en, fif.rd_en} == 2'b11) && fif.full)      // priority for write operation
            count <= count - 1;
        else if (({fif.wr_en, fif.rd_en} == 2'b11) && fif.empty)      // priority for read operation
            count <= count + 1;
    end end
assign fif.full = (count == fif.FIFO_DEPTH)? 1 : 0;
assign fif.empty = (count == 0)? 1 : 0;
assign fif.almostfull = (count == fif.FIFO_DEPTH-1)? 1 : 0;
assign fif.almostempty = (count == 1)? 1 : 0;
endmodule
```

SCOREBOARD:



```
package FIFO_scoreboard_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
import FIFO_seq_item_pkg::*;
class FIFO_scoreboard extends uvm_scoreboard;
    `uvm_component_utils(FIFO_scoreboard)
    uvm_analysis_export #(FIFO_seq_item) sb_export;
    uvm_tlm_analysis_fifo #(FIFO_seq_item) sb_fifo;
    FIFO_seq_item obj;
    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;
    int correct_count, error_count;
    bit [FIFO_WIDTH-1:0] queue[$];
    localparam max_fifo_addr = $clog2(FIFO_DEPTH);
    function void check_data(input FIFO_seq_item 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 queue = %p", $time, queue);
            correct_count++;
        end else begin
            error_count++;
            $display("At time = %0t , the outputs of the DUT don't match with the Golden model outputs", $time);
        end
    endfunction
function void 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 reference_model(FIFO_seq_item F_rm);
        fork
            begin
                if (!F_rm.rst_n) begin
                    wr_ack_ref = 0;
                    full_ref = 0;
                    almostfull_ref = 0;
                    overflow_ref = 0;
                    counter = 0;
                    queue.delete();
                end else if (F_rm.wr_en && counter < F_rm.FIFO_DEPTH) begin
                    queue.push_back(F_rm.data_in);
                    wr_ack_ref = 1;
                end else begin
                    wr_ack_ref = 0;
                    if (full_ref && F_rm.wr_en)
                        overflow_ref = 1;
                    else
                        overflow_ref = 0;
                end
            end
```

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

        if (!F_rm.rst_n) begin
            counter = 0;
        end else if ({F_rm.wr_en, F_rm.rd_en} == 2'b10 && !full_ref) begin
            counter = counter + 1;
        end else if ({F_rm.wr_en, F_rm.rd_en} == 2'b01 && !empty_ref) begin
            counter = counter - 1;
        end else if ({F_rm.wr_en, F_rm.rd_en} == 2'b11 && full_ref) begin
            counter = counter - 1;
        end else if ({F_rm.wr_en, F_rm.rd_en} == 2'b11 && empty_ref) begin
            counter = counter + 1;
        end
        flags();
endfunction
function new(string name = "FIFO_scoreboard", uvm_component parent = null);
            super.new(name,parent);
        endfunction
        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            sb_export=new("sb_export",this);
            sb_fifo=new("sb_fifo",this);
        endfunction
        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            sb_export.connect(sb_fifo.analysis_export);
        endfunction
        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                sb_fifo.get(obj);
                check_data(obj);
            end
        endtask
        function void report_phase(uvm_phase phase);
            super.report_phase(phase);
            `uvm_info("report_phase", $sformatf("At time %0t: Simulation Ends and Error Count=
, Correct Count=
%0d%0d", $time, error_count, correct_count), UVM_MEDIUM);
endfunction
endpackage
```

READ_ONLY_SEQUENCE:

```
package FIFO_read_only_sequence_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import FIFO_seq_item_pkg::*;

        class FIFO_read_only_sequence extends uvm_sequence #(FIFO_seq_item);
            `uvm_object_utils(FIFO_read_only_sequence)

            FIFO_seq_item seq_item;

            function new(string name = "FIFO_read_only_sequence");
                super.new(name);
            endfunction

            task body;
                repeat(10) begin
                    seq_item=FIFO_seq_item::type_id::create("seq_item");
                    start_item(seq_item);
                    assert(seq_item.randomize() with {rst_n==1; wr_en==0; rd_en==1;});
                    finish_item(seq_item);
                end
            endtask

        endclass

    endpackage
```

DRIVER:

```
package FIFO_driver_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
import FIFO_seq_item_pkg::*;

    class FIFO_driver extends uvm_driver #(FIFO_seq_item);
        `uvm_component_utils(FIFO_driver)

        virtual FIFO_if FIFO_vif;
        FIFO_seq_item stim_seq_item;

        function new(string name = "FIFO_driver", uvm_component parent = null);
            super.new(name,parent);
        endfunction

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                stim_seq_item=FIFO_seq_item::type_id::create("stim_seq_item");
                seq_item_port.get_next_item(stim_seq_item);
                FIFO_vif.data_in=stim_seq_item.data_in;
                FIFO_vif.rst_n=stim_seq_item.rst_n;
                FIFO_vif.wr_en=stim_seq_item.wr_en;
                FIFO_vif.rd_en=stim_seq_item.rd_en;
                @(negedge FIFO_vif.clk);
                seq_item_port.item_done();
            end
        endtask
    endclass
endpackage
```

ENVIROMENT:

```
package FIFO_env_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
import FIFO_agent_pkg::*;
import FIFO_scoreboard_pkg::*;
import FIFO_coverage_pkg::*;

    class FIFO_env extends uvm_env;
        `uvm_component_utils(FIFO_env)

        FIFO_agent agt;
        FIFO_scoreboard sb;
        FIFO_coverage cov;

        function new(string name = "FIFO_env", uvm_component parent = null);
            super.new(name,parent);
        endfunction

        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            agt=FIFO_agent::type_id::create("agt",this);
            sb=FIFO_scoreboard::type_id::create("sb",this);
            cov=FIFO_coverage::type_id::create("cov",this);
        endfunction

        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            agt.agt_ap.connect(sb.sb_export);
            agt.agt_ap.connect(cov.cov_export);
        endfunction
    endclass

endpackage
```

TEST:

```
    package FIFO_test_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
import FIFO_env_pkg::*;
import FIFO_reset_sequence_pkg::*;
import FIFO_write_only_sequence_pkg::*;
import FIFO_read_only_sequence_pkg::*;
import FIFO_write_read_sequence_pkg::*;
import FIFO_config_pkg::*;
    class FIFO_test extends uvm_component;
        `uvm_component_utils(FIFO_test)
        FIFO_env env;
        FIFO_reset_sequence reset_sequence;
        FIFO_write_only_sequence write_only_sequence;
        FIFO_read_only_sequence read_only_sequence;
        FIFO_write_read_sequence write_read_sequence;
        FIFO_config FIFO_cfg;
        function new(string name = "FIFO_test", uvm_component parent = null);
            super.new(name,parent);
        endfunction
        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            env=FIFO_env::type_id::create("env",this);
            reset_sequence=FIFO_reset_sequence::type_id::create("reset_sequence");
            write_only_sequence=FIFO_write_only_sequence::type_id::create("write_only_sequence");
            read_only_sequence=FIFO_read_only_sequence::type_id::create("read_only_sequence");
            write_read_sequence=FIFO_write_read_sequence::type_id::create("write_read_sequence");
            FIFO_cfg=FIFO_config::type_id::create("FIFO_cfg");
            if (!uvm_config_db#(virtual FIFO_if)::get(this, "", "FIFO_IF", FIFO_cfg.FIFO_vif)) begin
                `uvm_fatal("build_phase", "TEST - unable to get the IF");
            end
            uvm_config_db#(FIFO_config)::set(this, "*", "CFG", FIFO_cfg);
        endfunction
        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            phase.raise_objection(this);
            `uvm_info("run_phase", "Reset Asserted", UVM_LOW);
            reset_sequence.start(env.agt.sqr);
            `uvm_info("run_phase", "Reset De-asserted", UVM_LOW);
            `uvm_info("run_phase", "write_only_sequence starts", UVM_LOW);
            write_only_sequence.start(env.agt.sqr);
            `uvm_info("run_phase", "write_only_sequence ends", UVM_LOW);
            `uvm_info("run_phase", "read_only_sequence starts", UVM_LOW);
            read_only_sequence.start(env.agt.sqr);
            `uvm_info("run_phase", "read_only_sequence ends", UVM_LOW);
            `uvm_info("run_phase", "write_read_sequence starts", UVM_LOW);
            write_read_sequence.start(env.agt.sqr);
            `uvm_info("run_phase", "write_read_sequence ends", UVM_LOW);
            phase.drop_objection(this);
        endtask
    endclass
endpackage
```

TOP:

```
import uvm_pkg::*;
`include "uvm_macros.svh"
import FIFO_test_pkg::*;

module top;
    bit clk;
    initial begin
        clk=0;
        forever #1 clk=~clk;
    end
    FIFO_if fif(clk);
    FIFO dut(fif);

    bind FIFO FIFO_sva FIFO_sva_inst (fif, dut.count);

    initial begin
        uvm_config_db#(virtual FIFO_if)::set(null, "uvm_test_top", "FIFO_IF", fif);
        run_test("FIFO_test");
    end

endmodule
```

SVA:

```
module FIFO_sva(FIFO_if.DUT fif, input [fif.max_fifo_addr:0] count);
always_comb begin
    if(!fif.rst_n)
    reset_1_assertion: assert final ((!fif.wr_ack)&&(!fif.overflow)&&(!fif.underflow)&&(!count));
    reset_1_cover: cover final ((!fif.wr_ack)&&(!fif.overflow)&&(!fif.underflow)&&(!count));
  end
  always_comb begin
    if((fif.rst_n)&&(count == fif.FIFO_DEPTH))
    full_assertion: assert final (fif.full);
    full_cover: cover (fif.full);
  end
  always_comb begin
    if((fif.rst_n)&&(count == 0))
    empty_assertion: assert final (fif.empty);
    empty_cover: cover (fif.empty);
  end
  always_comb begin
    if((fif.rst_n)&&(count == fif.FIFO_DEPTH-1))
    almostfull_assertion: assert final (fif.almostfull);
    almostfull_cover: cover (fif.almostfull);
  end
  always_comb begin
    if((fif.rst_n)&&(count == 1))
    almostempty_assertion: assert final (fif.almostempty);
    almostempty_cover: cover (fif.almostempty);
  end
property P0;
    @(posedge fif.clk or negedge fif.rst_n)
    (!fif.rst_n) |-> ((!fif.wr_ack)&&(!fif.overflow)&&(!fif.underflow));
endproperty
reset_clk:assert property(P0);
reset_clk_cover:cover property(P0);
property p1;
    @(posedge fif.clk) disable iff(!fif.rst_n)
      (fif.wr_en && !fif.full) |=> fif.wr_ack;
endproperty
  write_acknowledge : assert property(p1);
  write_acknowledge_cover : cover property(p1);
property p2;
    @(posedge fif.clk) disable iff(!fif.rst_n)
      (fif.empty && fif.rd_en) |=> fif.underflow;
endproperty
  underflow_assertion : assert property(p2);
  underflow_cover : cover property(p2);
  property p3;
    @(posedge fif.clk) disable iff(!fif.rst_n)
      (fif.full && fif.wr_en) |=> fif.overflow;
endproperty
  overflow_assertion : assert property(p3);
  overflow_cover : cover property(p3);
property p4;
    @(posedge fif.clk) disable iff(!fif.rst_n)
      (!fif.full && fif.wr_en && !fif.rd_en) |=>
      (count == $past(count) + 4'b0001);
endproperty
  increment_assertion : assert property(p4);
  increment_cover : cover property(p4);
  property p5;
    @(posedge fif.clk) disable iff(!fif.rst_n)
      (!fif.empty && fif.rd_en && !fif.wr_en) |=>
      (count == $past(count) - 4'b0001);
  endproperty
  decrement_assertion : assert property(p5);
  decrement_cover : cover property(p5);
endmodule
```

INTERFACE:

```
interface FIFO_if (input bit clk);
    parameter FIFO_WIDTH = 16;
    parameter FIFO_DEPTH = 8;
    parameter max_fifo_addr = $clog2(FIFO_DEPTH);

    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
    );

endinterface
```

SEQUENCER:

```
package FIFO_sequencer_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
import FIFO_seq_item_pkg::*;

    class FIFO_sequencer extends uvm_sequencer #(FIFO_seq_item);
        `uvm_component_utils(FIFO_sequencer)

        function new(string name = "FIFO_sequencer", uvm_component parent = null);
            super.new(name,parent);
        endfunction

    endclass

endpackage
```

WRITE_SEQUENCE:

```
package FIFO_write_only_sequence_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
import FIFO_seq_item_pkg::*;

    class FIFO_write_only_sequence extends uvm_sequence #(FIFO_seq_item);
        `uvm_object_utils(FIFO_write_only_sequence)

        FIFO_seq_item seq_item;

        function new(string name = "FIFO_write_only_sequence");
            super.new(name);
        endfunction

        task body;
            repeat(10) begin
                seq_item=FIFO_seq_item::type_id::create("seq_item");
                start_item(seq_item);
                assert(seq_item.randomize() with {rst_n==1; wr_en==1; rd_en==0;});
                finish_item(seq_item);
            end
        endtask

    endclass

endpackage
```

CONFIG:

```
package FIFO_config_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"

    class FIFO_config extends  uvm_object;
        `uvm_object_utils(FIFO_config)

        virtual FIFO_if FIFO_vif;

        function new(string name = "FIFO_config");
            super.new(name);
        endfunction

    endclass

endpackage
```

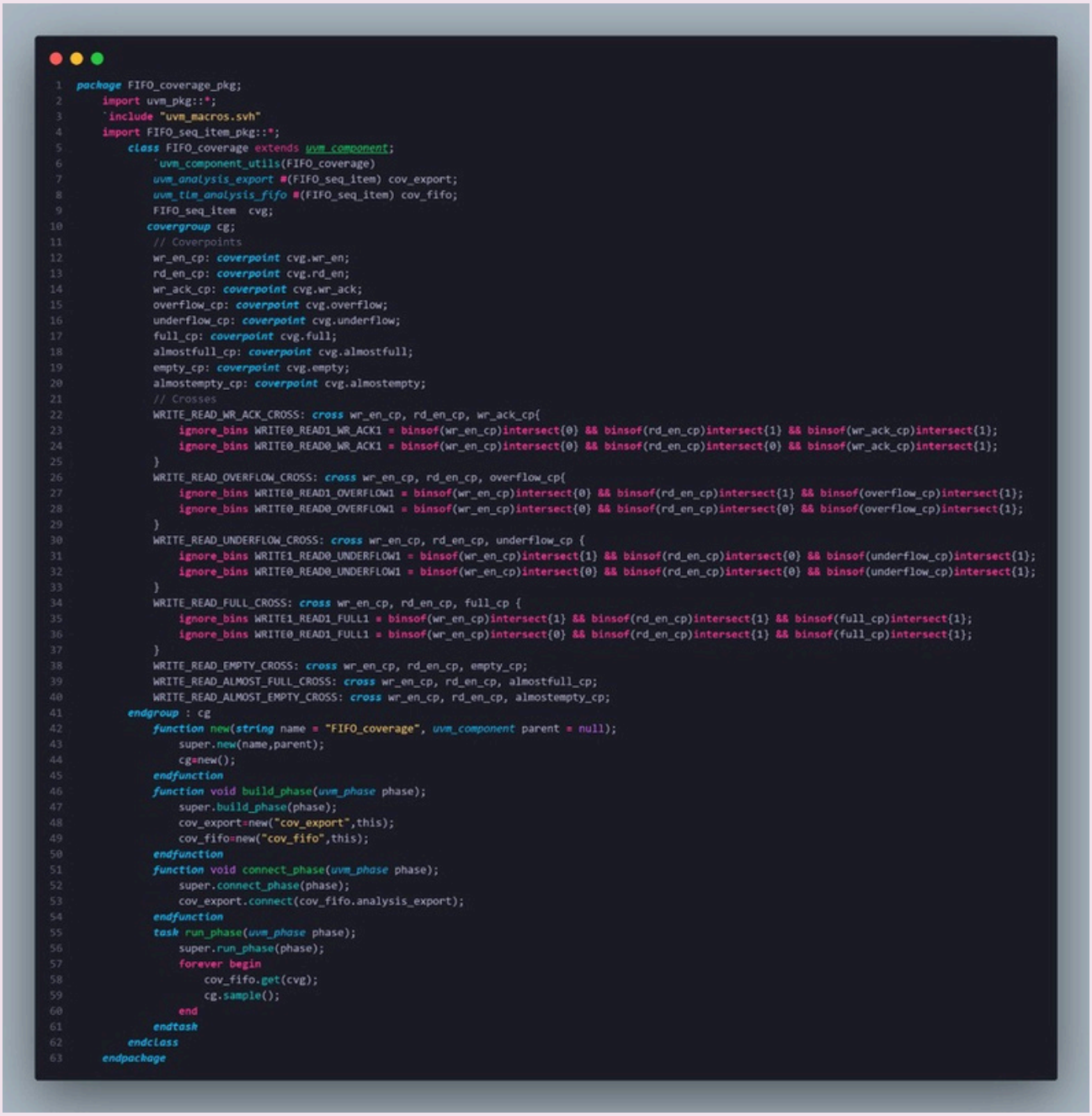
AGENT:

```
package FIFO_agent_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
import FIFO_sequencer_pkg::*;
import FIFO_driver_pkg::*;
import FIFO_monitor_pkg::*;
import FIFO_config_pkg::*;
import FIFO_seq_item_pkg::*;

    class FIFO_agent extends uvm_agent;
        `uvm_component_utils(FIFO_agent)

        FIFO_sequencer sqr;
        FIFO_driver drv;
        FIFO_monitor mon;
        FIFO_config FIFO_cfg;
        uvm_analysis_port #(FIFO_seq_item) agt_ap;

        function new(string name = "FIFO_agent", uvm_component parent = null);
            super.new(name,parent);
        endfunction

        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            sqr=FIFO_sequencer::type_id::create("sqr",this);
            drv=FIFO_driver::type_id::create("drv",this);
            mon=FIFO_monitor::type_id::create("mon",this);
            agt_ap=new("agt_ap",this);

            if ( !uvm_config_db#(FIFO_config)::get(this, "", "CFG", FIFO_cfg) ) begin
                `uvm_fatal("build_phase", "AGENT - unable to get the IF");
            end

        endfunction

        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            drv.FIFO_vif=FIFO_cfg.FIFO_vif;
            mon.FIFO_vif=FIFO_cfg.FIFO_vif;
            drv.seq_item_port.connect(sqr.seq_item_export);
            mon.mon_ap.connect(agt_ap);
        endfunction
    endclass

endpackage
```

SEQ_ITEM:

```
package FIFO_seq_item_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
        class FIFO_seq_item extends uvm_sequence_item;
            `uvm_object_utils(FIFO_seq_item)
            parameter FIFO_WIDTH = 16;
            parameter FIFO_DEPTH = 8;
            rand logic [FIFO_WIDTH-1:0] data_in;
            rand logic rst_n, wr_en, rd_en;
            logic [FIFO_WIDTH-1:0] data_out;
            logic wr_ack, overflow,underflow;
            logic full, empty, almostfull, almostempty;
            int WR_EN_ON_DIST= 70;
            int RD_EN_ON_DIST= 30;
            constraint rst_c { rst_n dist {0:=2, 1:=98}; }
            constraint wr_en_c { wr_en dist {1:=WR_EN_ON_DIST, 0:=100-WR_EN_ON_DIST}; }
            constraint rd_en_c { rd_en dist {1:=RD_EN_ON_DIST, 0:=100-RD_EN_ON_DIST}; }
            function new(string name = "FIFO_seq_item");
                super.new(name);
            endfunction

            function string convert2string();
             return $sformatf("%s reset=0b%0b, wr_en=0b%0b, rd_en=0b%0b, data_in=0x%0x,data_out=0x%0x, full=0b%0b, empty=0b%0b", super.convert2string(),
             rst_n, wr_en, rd_en, data_in,data_out, full, empty);
            endfunction

            function string convert2string_stimulus();
            return $sformatf("reset=0b%0b, wr_en=0b%0b, rd_en=0b%0b, data_in=0x%0x",rst_n, wr_en, rd_en, data_in);
            endfunction
        endclass
    endpackage
```

COVERAGE:



```
package FIFO_coverage_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    import FIFO_seq_item_pkg::*;
        class FIFO_coverage extends uvm_component;
            `uvm_component_utils(FIFO_coverage)
            uvm_analysis_export #(FIFO_seq_item) cov_export;
            uvm_tlm_analysis_fifo #(FIFO_seq_item) cov_fifo;
            FIFO_seq_item  cvg;
           covergroup cg;
            // Coverpoints
            wr_en_cp: coverpoint cvg.wr_en;
            rd_en_cp: coverpoint cvg.rd_en;
            wr_ack_cp: coverpoint cvg.wr_ack;
            overflow_cp: coverpoint cvg.overflow;
            underflow_cp: coverpoint cvg.underflow;
            full_cp: coverpoint cvg.full;
            almostfull_cp: coverpoint cvg.almostfull;
            empty_cp: coverpoint cvg.empty;
            almostempty_cp: coverpoint cvg.almostempty;
            // Crosses
            WRITE_READ_WR_ACK_CROSS: cross wr_en_cp, rd_en_cp, wr_ack_cp{
                ignore_bins WRITE0_READ1_WR_ACK1 = binsof(wr_en_cp)intersect{0} && binsof(rd_en_cp)intersect{1} && binsof(wr_ack_cp)intersect{1};
                ignore_bins WRITE0_READ0_WR_ACK1 = binsof(wr_en_cp)intersect{0} && binsof(rd_en_cp)intersect{0} && binsof(wr_ack_cp)intersect{1};
            }
            WRITE_READ_OVERFLOW_CROSS: cross wr_en_cp, rd_en_cp, overflow_cp{
                ignore_bins WRITE0_READ1_OVERFLOW1 = binsof(wr_en_cp)intersect{0} && binsof(rd_en_cp)intersect{1} && binsof(overflow_cp)intersect{1};
                ignore_bins WRITE0_READ0_OVERFLOW1 = binsof(wr_en_cp)intersect{0} && binsof(rd_en_cp)intersect{0} && binsof(overflow_cp)intersect{1};
            }
            WRITE_READ_UNDERFLOW_CROSS: cross wr_en_cp, rd_en_cp, underflow_cp {
                ignore_bins WRITE1_READ0_UNDERFLOW1 = binsof(wr_en_cp)intersect{1} && binsof(rd_en_cp)intersect{0} && binsof(underflow_cp)intersect{1};
                ignore_bins WRITE0_READ0_UNDERFLOW1 = binsof(wr_en_cp)intersect{0} && binsof(rd_en_cp)intersect{0} && binsof(underflow_cp)intersect{1};
            }
            WRITE_READ_FULL_CROSS: cross wr_en_cp, rd_en_cp, full_cp {
                ignore_bins WRITE1_READ1_FULL1 = binsof(wr_en_cp)intersect{1} && binsof(rd_en_cp)intersect{1} && binsof(full_cp)intersect{1};
                ignore_bins WRITE0_READ1_FULL1 = binsof(wr_en_cp)intersect{0} && binsof(rd_en_cp)intersect{1} && binsof(full_cp)intersect{1};
            }
            WRITE_READ_EMPTY_CROSS: cross wr_en_cp, rd_en_cp, empty_cp;
            WRITE_READ_ALMOST_FULL_CROSS: cross wr_en_cp, rd_en_cp, almostfull_cp;
            WRITE_READ_ALMOST_EMPTY_CROSS: cross wr_en_cp, rd_en_cp, almostempty_cp;
        endgroup : cg
            function new(string name = "FIFO_coverage", uvm_component parent = null);
                super.new(name,parent);
                cg=new();
            endfunction
            function void build_phase(uvm_phase phase);
                super.build_phase(phase);
                cov_export=new("cov_export",this);
                cov_fifo=new("cov_fifo",this);
            endfunction
            function void connect_phase(uvm_phase phase);
                super.connect_phase(phase);
                cov_export.connect(cov_fifo.analysis_export);
            endfunction
            task run_phase(uvm_phase phase);
                super.run_phase(phase);
                forever begin
                    cov_fifo.get(cvg);
                    cg.sample();
                end
            endtask
        endclass
    endpackage
```

WRITE_READ_SEQ:

```
package FIFO_write_read_sequence_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
import FIFO_seq_item_pkg::*;

    class FIFO_write_read_sequence extends uvm_sequence #(FIFO_seq_item);
        `uvm_object_utils(FIFO_write_read_sequence)

        FIFO_seq_item seq_item;

        function new(string name = "FIFO_write_read_sequence");
            super.new(name);
        endfunction

        task body;
            repeat(10000) begin
                seq_item=FIFO_seq_item::type_id::create("seq_item");
                start_item(seq_item);
                assert(seq_item.randomize());
                finish_item(seq_item);
            end
        endtask

    endclass

endpackage
```

RESET_SEQUENCE:

```
package FIFO_reset_sequence_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
import FIFO_seq_item_pkg::*;

    class FIFO_reset_sequence extends uvm_sequence #(FIFO_seq_item);
        `uvm_object_utils(FIFO_reset_sequence)

        FIFO_seq_item seq_item;

        function new(string name = "FIFO_reset_sequence");
            super.new(name);
        endfunction

        task body;
            seq_item=FIFO_seq_item::type_id::create("seq_item");
            start_item(seq_item);
            seq_item.rst_n=0;
            finish_item(seq_item);
        endtask

    endclass

endpackage
```

MONITOR:

```
package FIFO_monitor_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
import FIFO_seq_item_pkg::*;

    class FIFO_monitor extends uvm_monitor;
        `uvm_component_utils(FIFO_monitor)

        virtual FIFO_if FIFO_vif;
        FIFO_seq_item rsp_seq_item;

        uvm_analysis_port #(FIFO_seq_item) mon_ap;

        function new(string name = "FIFO_monitor", uvm_component parent = null);
            super.new(name,parent);
        endfunction

        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            mon_ap=new("mon_ap",this);
        endfunction

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                rsp_seq_item=FIFO_seq_item::type_id::create("rsp_seq_item");
                @(negedge FIFO_vif.clk);
                // outputs
                rsp_seq_item.data_out=FIFO_vif.data_out;
                rsp_seq_item.wr_ack=FIFO_vif.wr_ack;
                rsp_seq_item.overflow=FIFO_vif.overflow;
                rsp_seq_item.full=FIFO_vif.full;
                rsp_seq_item.empty=FIFO_vif.empty;
                rsp_seq_item.almostfull=FIFO_vif.almostfull;
                rsp_seq_item.almostempty=FIFO_vif.almostempty;
                rsp_seq_item.underflow=FIFO_vif.underflow;
                // inputs
                rsp_seq_item.data_in=FIFO_vif.data_in;
                rsp_seq_item.wr_en=FIFO_vif.wr_en;
                rsp_seq_item.rd_en=FIFO_vif.rd_en;
                rsp_seq_item.rst_n=FIFO_vif.rst_n;
                mon_ap.write(rsp_seq_item);
                `uvm_info("run_phase", rsp_seq_item.convert2string(), UVM_HIGH);
            end
        endtask

    endclass
endpackage
```

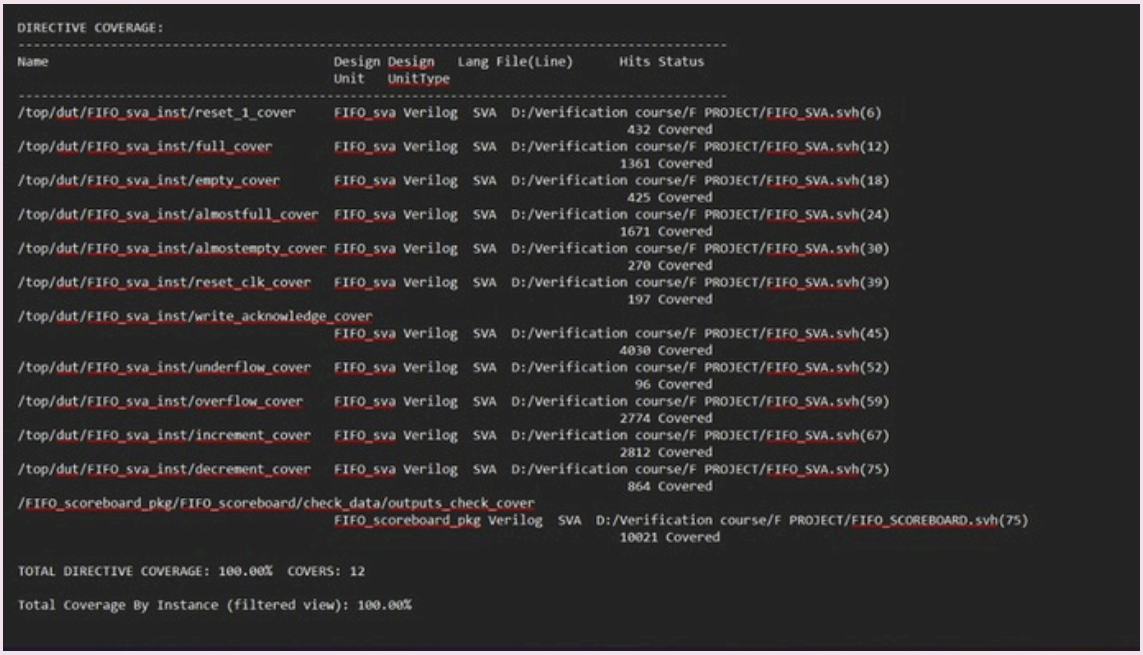
CODE COVERAGE:

```
Coverage Report Summary Data by DU

================================================================================
=== Design Unit: work.FIFO
================================================================================
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        25        25         0   100.00%
    Conditions                      24        24         0   100.00%
    Statements                      28        28         0   100.00%
    Toggles                         20        20         0   100.00%

Total Coverage By Design Unit (filtered view): 100.00%
```

RUN DO:

```
vlib work
vlog -f src_files.list -mfcu +cover -covercells
vsim -voptargs=+acc work.top -cover -classdebug -uvmcontrol=all
add wave /top/fif/*
coverage save top.ucdb -onexit
coverage exclude -src FIFO.svh -line 22 -code c
coverage exclude -src FIFO.svh -line 40 -code c
run -all
```

FUNCTIONAL COVERAGE:



```
DIRECTIVE COVERAGE:
--------------------------------------------------------------------------------------------
Name                                    Design Design   Lang File(Line)      Hits Status
                                        Unit   UnitType
--------------------------------------------------------------------------------------------
/top/dut/FIFO_sva_inst/reset_1_cover    FIFO_sva Verilog  SVA  D:/Verification course/F PROJECT/FIFO_SVA.svh(6)
                                                                                 432 Covered
/top/dut/FIFO_sva_inst/full_cover       FIFO_sva Verilog  SVA  D:/Verification course/F PROJECT/FIFO_SVA.svh(12)
                                                                                1361 Covered
/top/dut/FIFO_sva_inst/empty_cover      FIFO_sva Verilog  SVA  D:/Verification course/F PROJECT/FIFO_SVA.svh(18)
                                                                                 425 Covered
/top/dut/FIFO_sva_inst/almostfull_cover FIFO_sva Verilog  SVA  D:/Verification course/F PROJECT/FIFO_SVA.svh(24)
                                                                                1671 Covered
/top/dut/FIFO_sva_inst/almostempty_cover FIFO_sva Verilog  SVA  D:/Verification course/F PROJECT/FIFO_SVA.svh(30)
                                                                                 270 Covered
/top/dut/FIFO_sva_inst/reset_clk_cover  FIFO_sva Verilog  SVA  D:/Verification course/F PROJECT/FIFO_SVA.svh(39)
                                                                                 197 Covered
/top/dut/FIFO_sva_inst/write_acknowledge_cover
                                        FIFO_sva Verilog  SVA  D:/Verification course/F PROJECT/FIFO_SVA.svh(45)
                                                                                4030 Covered
/top/dut/FIFO_sva_inst/underflow_cover  FIFO_sva Verilog  SVA  D:/Verification course/F PROJECT/FIFO_SVA.svh(52)
                                                                                  96 Covered
/top/dut/FIFO_sva_inst/overflow_cover   FIFO_sva Verilog  SVA  D:/Verification course/F PROJECT/FIFO_SVA.svh(59)
                                                                                2774 Covered
/top/dut/FIFO_sva_inst/increment_cover  FIFO_sva Verilog  SVA  D:/Verification course/F PROJECT/FIFO_SVA.svh(67)
                                                                                2812 Covered
/top/dut/FIFO_sva_inst/decrement_cover  FIFO_sva Verilog  SVA  D:/Verification course/F PROJECT/FIFO_SVA.svh(75)
                                                                                 864 Covered
/FIFO_scoreboard_pkg/FIFO_scoreboard/check_data/outputs_check_cover
                                        FIFO_scoreboard_pkg Verilog  SVA  D:/Verification course/F PROJECT/FIFO_SCOREBOARD.svh(75)
                                                                               10021 Covered

TOTAL DIRECTIVE COVERAGE: 100.00%  COVERS: 12

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

SRC:

```
FIFO.svh
FIFO_INTERFACE.svh
FIFO_CONFIG.svh
FIFO_SEQ_ITEM.svh
FIFO_DRIVER.svh
FIFO_MONITOR.svh
FIFO_SEQUENCER.svh
FIFO_AGENT.svh
FIFO_COVERAGE_COLLECTOR.svh
FIFO_SCOREBOARD.svh
FIFO_ENV.svh
FIFO_RD_ONLY.svh
FIFO_RST.svh
FIFO_WR_ONLY.svh
FIFO_WR_RD.svh
FIFO_TEST.svh
FIFO_SVA.svh
FIFO_TOP.svh
```

ASSERTIONS:

Feature	Assertion
FIFO reset clears wr_ack, overflow, and underflow	@(posedge fif.clk or negedge fif.rst_n) (!fif.rst_n)
FIFO is full	@(posedge fif.clk) disable iff(!fif.rst_n) (count == fif.FIFO_DEPTH)
FIFO is empty	@(posedge fif.clk) disable iff(!fif.rst_n) (count == 0)
FIFO is almost full	@(posedge fif.clk) disable iff(!fif.rst_n) (count == fif.FIFO_DEPTH-1)
FIFO is almost empty	@(posedge fif.clk) disable iff(!fif.rst_n) (count == 1)
Write acknowledge when write enabled and not full	@(posedge fif.clk) disable iff(!fif.rst_n) (fif.wr_en && !fif.full)
Underflow occurs on read when FIFO empty	@(posedge fif.clk) disable iff(!fif.rst_n) (fif.empty && fif.rd_en)
Overflow occurs on write when FIFO full	@(posedge fif.clk) disable iff(!fif.rst_n) (fif.full && fif.wr_en)
Increment count on write when not full and no read	@(posedge fif.clk) disable iff(!fif.rst_n) (!fif.full && fif.wr_en && !fif.rd_en)
Decrement count on read when not empty and no write	@(posedge fif.clk) disable iff(!fif.rst_n) (!fif.empty && fif.rd_en && !fif.wr_en)

BUG REPORT:

	Description	Expected Behavior	Correction
1	[illegible] overflow, wr_ack, and underflow are not properly reset.	During reset, overflow, wr_ack, and underflow signals should be set to 0.	Ensure overflow, wr_ack, and underflow signals are explicitly reset to 0 in the reset handling block. data_out should remain unaffected by reset.
2	Unhandled simultaneous read_enable and write_enable cases:	Case 1: If FIFO is empty, only write should take place. Case 2: If FIFO is full, only read should occur.	Update the FIFO model to check full and empty flags when both read_enable and write_enable are high, prioritizing the following: - If empty: Execute only write. - If full: Execute only read.
3	underflow signal is treated as combinational but should be sequential.	underflow should be set on the next clock cycle if a read operation occurs on an empty FIFO.	Implement underflow as a sequential signal by adding a clocked process, so it activates only on the following clock cycle when read_enable is asserted while FIFO is empty.
4	almostfull flag depth condition is incorrect, defined as FIFO_DEPTH - 2 instead of FIFO_DEPTH - 1	almostfull should trigger when the FIFO reaches a count of FIFO_DEPTH - 1.	Modify the almostfull flag condition to activate at FIFO_DEPTH - 1 instead of FIFO_DEPTH - 2, ensuring it reflects the FIFO's near-full status accurately.

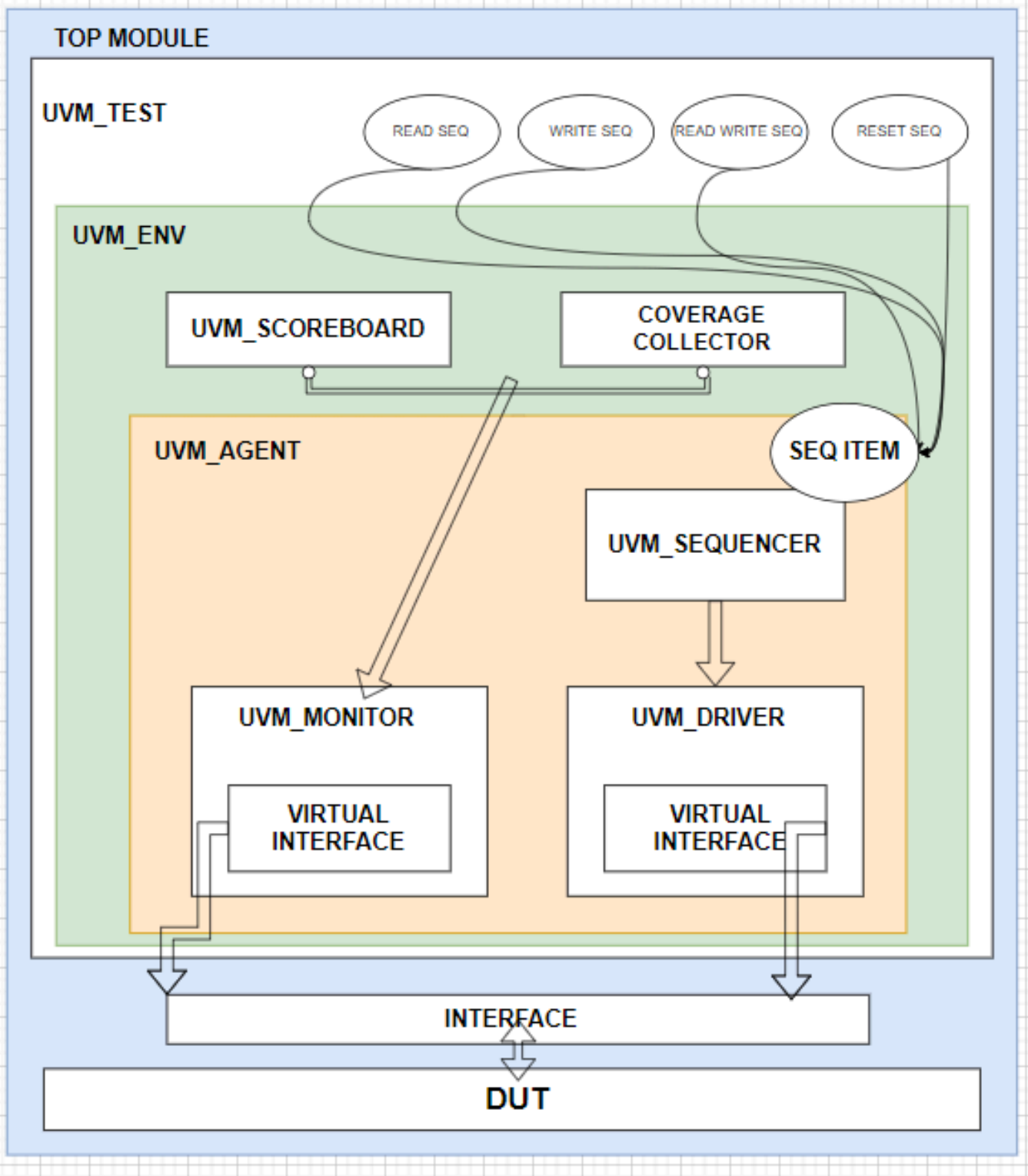
VERIFICATION PLAN:

Label	Description	Stimulus Generation	Functional Coverage	Functionality Check
FIFO_1	empty should not be HIGH if write enable is HIGH.	Randomization on wr_en under constraint that write occurs more than read.	Included as cross cover for wr_en and empty.	Output checked against golden model and also with assertion.
FIFO_2	full should not be HIGH if read enable is HIGH	Randomization on rd_en under constraint that write occurs more than read.	Included as cross cover for rd_en and full.	Output checked against golden model and also with assertion.
FIFO_3	overflow should not be HIGH if write enable is LOW	Randomization on wr_en under constraint that write occurs more than read.	Included as cross cover for wr_en and overflow.	Output checked against golden model and also with assertion.
FIFO_4	underflow should not be HIGH if read enable is LOW	Randomization on rd_en under constraint that write occurs more than read.	Included as cross cover for wr_en and underflow	Output checked against golden model and also with assertion.
FIFO_5	write ack should not be HIGH if write enable is LOW.	Randomization on wr_en under constraint that write occurs more than read.	Included as cross cover for wr_en and wr_ack.	Output checked against golden model and also with assertion.
FIFO_6	If both read enable and write enable are HIGH and FIFO is full, only read should occur.	Randomization on wr_en and rd_en under constraint that write occurs more than read.	Included as cross cover for wr_en, rd_en, and all output flags.	Output checked against golden model and also with assertion.
FIFO_7	If both read enable and write enable are HIGH and FIFO is empty, only write should occur	Randomization on wr_en and rd_en under constraint that write occurs more than read.	Included as cross cover for wr_en, rd_en, and all output flags	Output checked against golden model and also with assertion.
FIFO_8	If both read enable and write enable are HIGH and FIFO is neither full nor empty, both should occur.	Randomization on wr_en and rd_en under constraint that write occurs more than read.	Included as cross cover for wr_en, rd_en, and all output flags.	Output checked against golden model and also with assertion.
FIFO_9	If reset is asserted, all output flags should be LOW.	Randomization on rst_n with constraint that reset is deasserted most of the time	Not included.	Output checked against golden model and also with assertion.

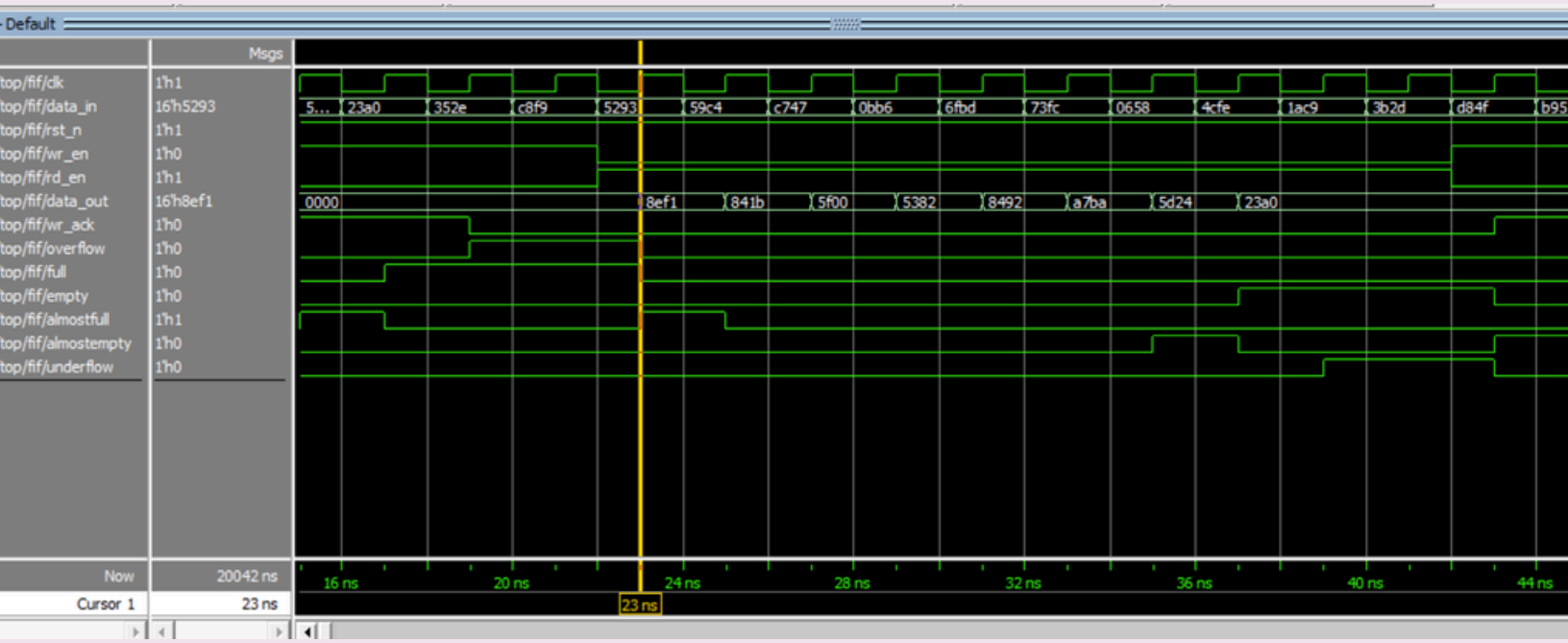
PRINT:

```
#
#   ************       IMPORTANT RELEASE NOTES         ************
#
#   You are using a version of the UVM library that has been compiled
#   with `UVM_NO_DEPRECATED undefined.
#   See http://www.eda.org/svdb/view.php?id=3313 for more details.
#
#   You are using a version of the UVM library that has been compiled
#   with `UVM_OBJECT_MUST_HAVE_CONSTRUCTOR undefined.
#   See http://www.eda.org/svdb/view.php?id=3770 for more details.
#
#       (Specify +UVM_NO_RELNOTES to turn off this notice)
#
# UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(277) @ 0: reporter [Questa UVM] QUESTA_UVM-1.2.3
# UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(278) @ 0: reporter [Questa UVM]  questa_uvm::init(+struct)
# UVM_INFO @ 0: reporter [RNTST] Running test FIFO_test...
# UVM_INFO D:/Verification course/F PROJECT/FIFO_TEST.svh(37) @ 0: uvm_test_top [run_phase] Reset Asserted
# UVM_INFO D:/Verification course/F PROJECT/FIFO_TEST.svh(39) @ 2: uvm_test_top [run_phase] Reset De-asserted
# UVM_INFO D:/Verification course/F PROJECT/FIFO_TEST.svh(40) @ 2: uvm_test_top [run_phase] write_only_sequence starts
# UVM_INFO D:/Verification course/F PROJECT/FIFO_TEST.svh(42) @ 22: uvm_test_top [run_phase] write_only_sequence ends
# UVM_INFO D:/Verification course/F PROJECT/FIFO_TEST.svh(43) @ 22: uvm_test_top [run_phase] read_only_sequence starts
# UVM_INFO D:/Verification course/F PROJECT/FIFO_TEST.svh(45) @ 42: uvm_test_top [run_phase] read_only_sequence ends
# UVM_INFO D:/Verification course/F PROJECT/FIFO_TEST.svh(46) @ 42: uvm_test_top [run_phase] write_read_sequence starts
# UVM_INFO D:/Verification course/F PROJECT/FIFO_TEST.svh(48) @ 20042: uvm_test_top [run_phase] write_read_sequence ends
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 20042: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO D:/Verification course/F PROJECT/FIFO_SCOREBOARD.svh(39) @ 20042: uvm_test_top.env.sb [report_phase] At time 20042: Simulation Ends and Error Count= 0, Correct Count= 10021
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   13
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# [report_phase]     1
# [run_phase]     8
# ** Note: $finish    : C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 20042 ns  Iteration: 61  Instance: /top
```

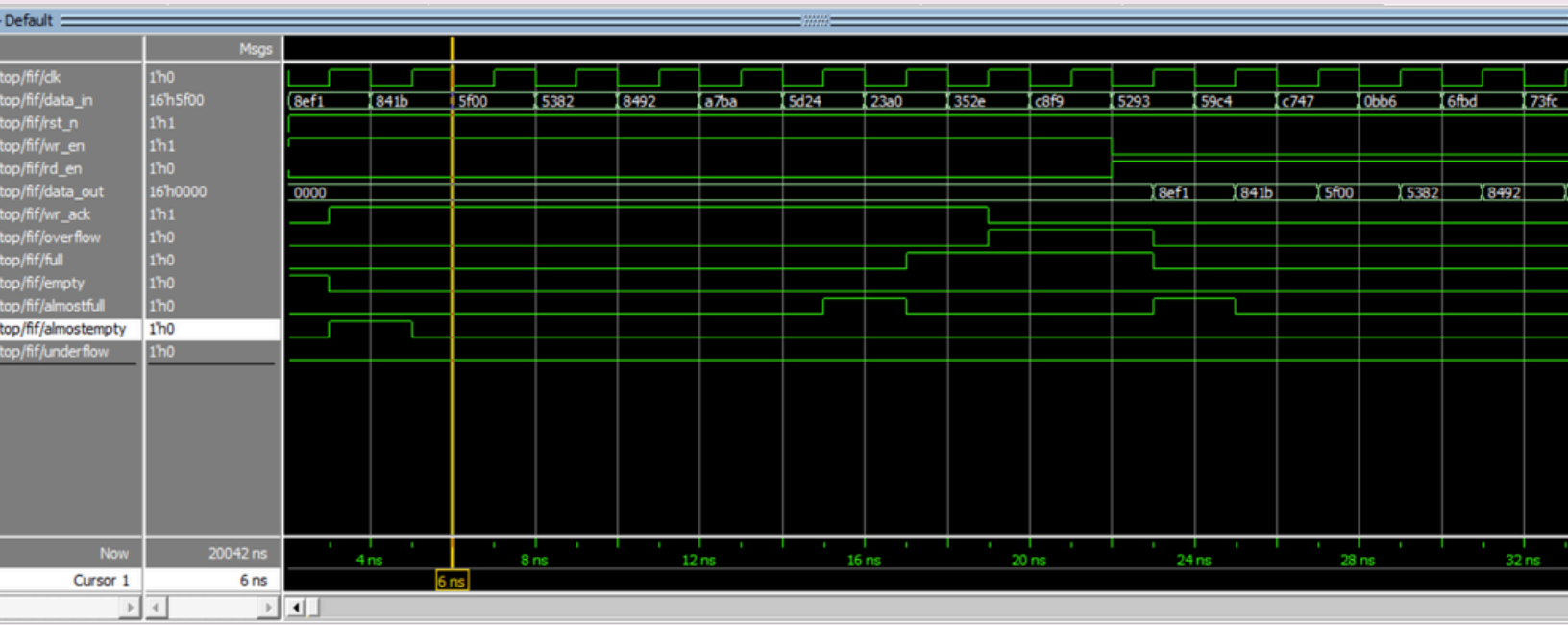
UVM Block :


TOP MODULE
UVM_TEST
READ SEQ
WRITE SEQ
READ WRITE SEQ
RESET SEQ
UVM_ENV
UVM_SCOREBOARD
COVERAGE COLLECTOR
UVM_AGENT
SEQ ITEM
UVM_SEQUENCER
UVM_MONITOR
VIRTUAL INTERFACE
UVM_DRIVER
VIRTUAL INTERFACE
INTERFACE
DUT

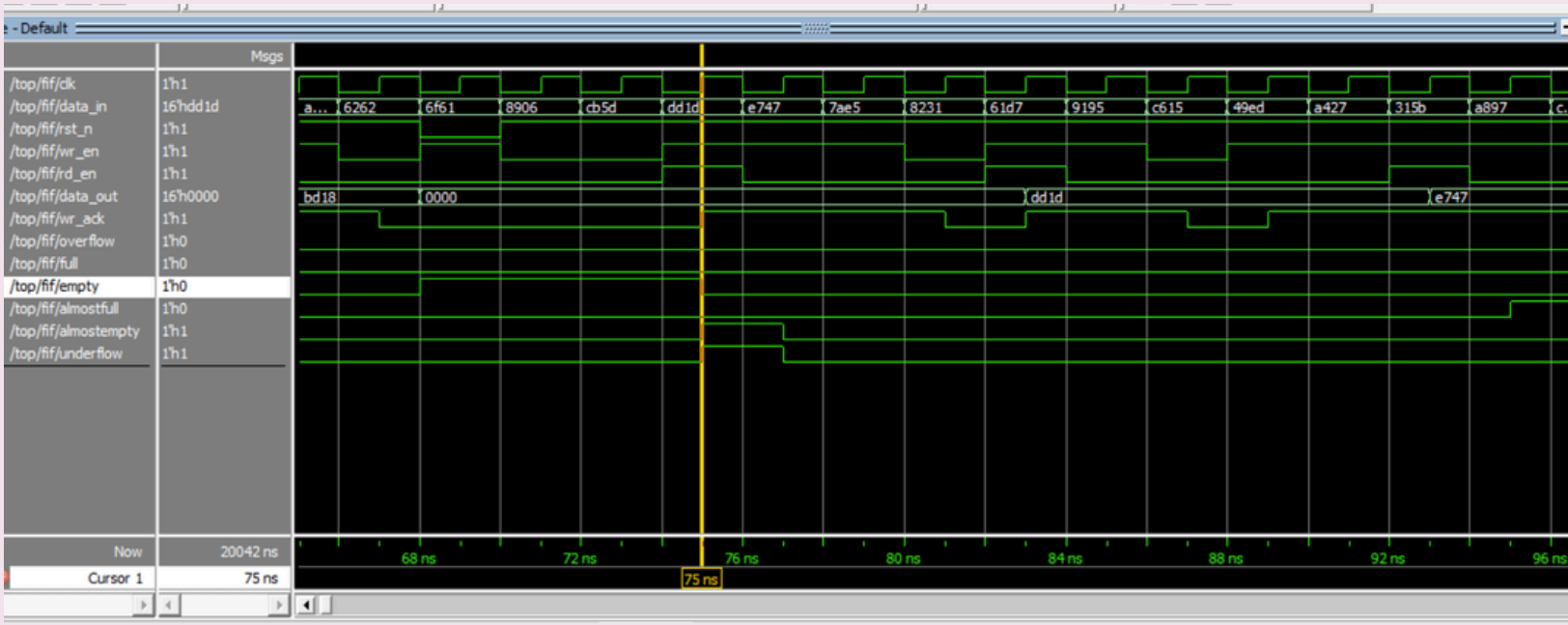
Read Sequence in Waveform :



Write Sequence in Waveform :



Write Read Sequence in Waveform :



Reset Sequence in Waveform :