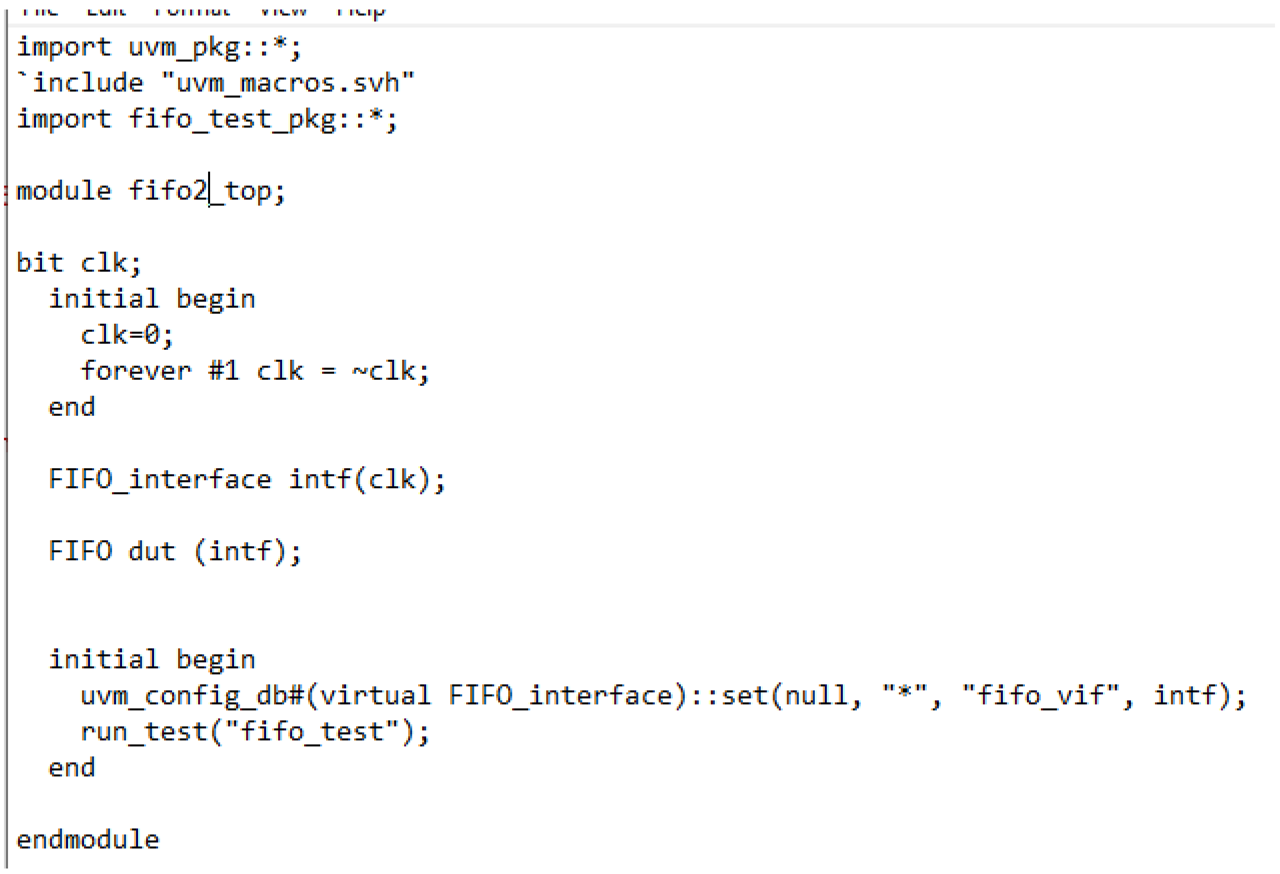
Name: Yousef Wael Mahmoud Alkattan

1. Top module



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

module fifo2_top;

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

  FIFO_interface intf(clk);

  FIFO dut (intf);


  initial begin
    uvm_config_db#(virtual FIFO_interface)::set(null, "*", "fifo_vif", intf);
    run_test("fifo_test");
  end

endmodule
```

2. Fixed Design & Assertions

```
module FIFO(FIFO_interface.dut intf);

        parameter FIFO_WIDTH = 16;
        parameter FIFO_DEPTH = 8;

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 intf.clk or negedge intf.rst_n) begin
        if (!intf.rst_n) begin
                wr_ptr <= 0;
                intf.wr_ack <= 0; //fixed
                intf.overflow <= 0; //fixed
        end
        else if (intf.wr_en && count < FIFO_DEPTH) begin
                mem[wr_ptr] <= intf.data_in;
                intf.wr_ack <= 1;
                wr_ptr <= wr_ptr + 1;
        end
        else begin
                intf.wr_ack <= 0;
                if (intf.full && intf.wr_en) //fixed was & only
                        intf.overflow <= 1;
                else
                        intf.overflow <= 0;
        end
end

always @(posedge intf.clk or negedge intf.rst_n) begin
        if (!intf.rst_n) begin
                rd_ptr <= 0;
                intf.underflow <= 0; //fixed
                intf.data_out <= 0;  //fixed
        end
        else if (intf.rd_en && count != 0) begin
                intf.data_out <= mem[rd_ptr];
                rd_ptr <= rd_ptr + 1;
        end
        else begin  //fixed underflow is sequential logic
                if (intf.empty && intf.rd_en)
                        intf.underflow <= 1;
                else
                        intf.underflow <= 0;
        end
end
```

```
always @(posedge intf.clk or negedge intf.rst_n) begin
        if (!intf.rst_n) begin
                count <= 0;
        end
        else begin
                if (({intf.wr_en, intf.rd_en} == 2'b11) && intf.full) //fixed fine if both
                        count <= count - 1;
                else if (({intf.wr_en, intf.rd_en} == 2'b11) && intf.empty) //fixed fine i
                        count <= count + 1;
                else if ( ({intf.wr_en, intf.rd_en} == 2'b10) && !intf.full)
                        count <= count + 1;
                else if ( ({intf.wr_en, intf.rd_en} == 2'b01) && !intf.empty)
                        count <= count - 1;
        end
end

assign intf.full = (count == FIFO_DEPTH)? 1 : 0;
assign intf.empty = (count == 0 && intf.rst_n)? 1 : 0; //fixed

assign intf.almostfull = (count == FIFO_DEPTH-1)? 1 : 0; //fixed -1 not -2
assign intf.almostempty = (count == 1)? 1 : 0;


//assertions

`ifdef SIM

// 1. Reset Behavior
always_comb begin : rst_n_assert
    if (!intf.rst_n) begin
        assert final (count == 0);
        assert final (wr_ptr == 0);
        assert final (rd_ptr == 0);

        assert final (intf.wr_ack == 0);
        assert final (intf.overflow == 0);
        assert final (intf.underflow == 0);
        assert final (intf.data_out == 0);

        assert final (intf.full == 0);
        assert final (intf.empty == 0);
        assert final (intf.almostfull == 0);
        assert final (intf.almostempty == 0);
    end
end
```

```
// 2. Write Acknowledge
ack: assert property (@(posedge intf.clk) disable iff(!intf.rst_n)
    (intf.wr_en && !intf.full) |=> intf.wr_ack);

// 3. Overflow Detection
overflow: assert property (@(posedge intf.clk) disable iff(!intf.rst_n)
    (intf.full && intf.wr_en) |=> intf.overflow);

// 4. Underflow Detection
underflow: assert property (@(posedge intf.clk) disable iff(!intf.rst_n)
    (intf.empty && intf.rd_en) |=> intf.underflow);

// 5. Empty Flag Assertion
empty: assert property (@(posedge intf.clk) disable iff(!intf.rst_n)
    (count == 0) |-> intf.empty);

// 6. Full Flag Assertion
full: assert property (@(posedge intf.clk) disable iff(!intf.rst_n)
    (count == FIFO_DEPTH) |-> intf.full);

// 7. Almost Full Condition
almostfull: assert property (@(posedge intf.clk) disable iff(!intf.rst_n)
    (count == FIFO_DEPTH-1) |-> intf.almostfull);

// 8. Almost Empty Condition
almostempty: assert property (@(posedge intf.clk) disable iff(!intf.rst_n)
    (count == 1) |-> intf.almostempty);

// 9. Pointer Wraparound
wr_ptr_assert: assert property (@(posedge intf.clk) disable iff(!intf.rst_n)
    (intf.wr_en && count < FIFO_DEPTH) |=>
    (wr_ptr == 0 ? $past(wr_ptr) == FIFO_DEPTH-1 : wr_ptr == $past(wr_ptr) + 1));

rd_ptr_assert: assert property (@(posedge intf.clk) disable iff(!intf.rst_n)
    (intf.rd_en && count != 0) |=>
    (rd_ptr == 0 ? $past(rd_ptr) == FIFO_DEPTH-1 : rd_ptr == $past(rd_ptr) + 1));

// 10. Pointer Threshold
count_range: assert property (@(posedge intf.clk) disable iff(!intf.rst_n)
    (count <= FIFO_DEPTH));

wr_ptr_range: assert property (@(posedge intf.clk) disable iff(!intf.rst_n)
    (wr_ptr < FIFO_DEPTH));

rd_ptr_range: assert property (@(posedge intf.clk) disable iff(!intf.rst_n)
    (rd_ptr < FIFO_DEPTH));

`endif


endmodule
```

3. Interface

```
interface FIFO_interface (clk);

    parameter FIFO_WIDTH = 16;
    parameter FIFO_DEPTH = 8;
    input bit clk;

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

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

    modport dut (
        input clk,
        input data_in, rst_n, wr_en, rd_en,

        output data_out,
        output wr_ack, overflow,
        output full, empty, almostfull, almostempty, underflow
    );


    modport tb (
        input clk,
        input data_out,
        input wr_ack, overflow,
        input full, empty, almostfull, almostempty, underflow,

        output data_in, rst_n, wr_en, rd_en
    );

    modport moniter (
        input clk,
        input data_in, rst_n, wr_en, rd_en,

        input data_out,
        input wr_ack, overflow,
        input full, empty, almostfull, almostempty, underflow

    );
endinterface
```

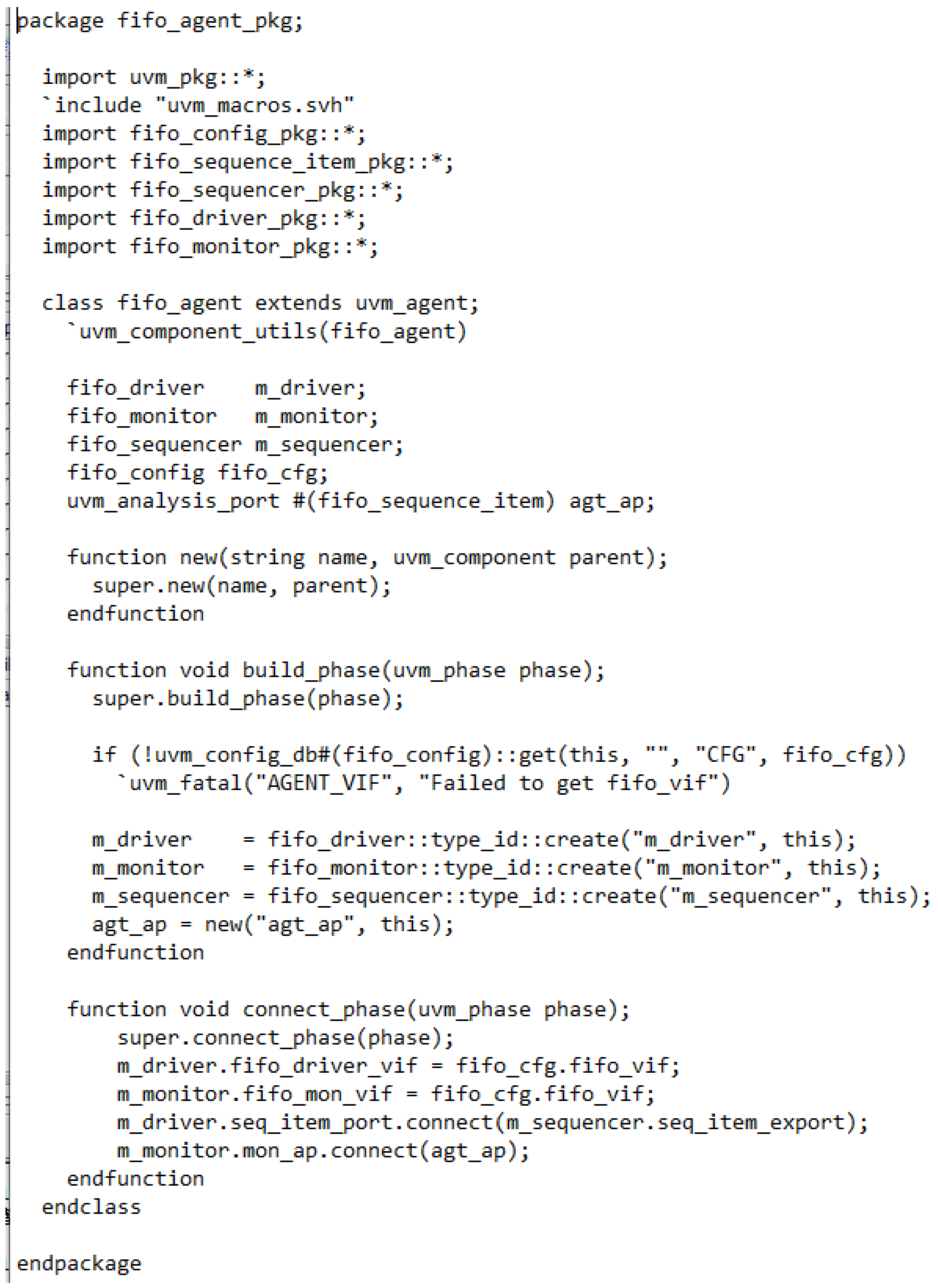
4. Test

```
package fifo_test_pkg;

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

import fifo_env_pkg::*;
import fifo_config_pkg::*;
import fifo_sequence_pkg::*;
import fifo_reset_sequence_pkg::*;

class fifo_test extends uvm_test;
   `uvm_component_utils(fifo_test)

    fifo_env env;
    fifo_config fifo_cfg;
    fifo_sequence seq1;
    fifo_reset_sequence seq2;

    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);
        fifo_cfg = fifo_config::type_id::create("fifo_cfg");
        seq1 = fifo_sequence::type_id::create("seq1");
        seq2 = fifo_reset_sequence::type_id::create("seq2");

        if (!uvm_config_db#(virtual FIFO_interface)::get(this, "", "fifo_vif", fifo_cfg.fifo_vif))
        `uvm_fatal("VIF_GET", "Failed to get virtual interface in fifo_test")

        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("FIFO_TEST", "Inside the Fifo test", UVM_MEDIUM)

        seq2.start(env.m_agent.m_sequencer);
        seq1.start(env.m_agent.m_sequencer);

        phase.drop_objection(this);
    endtask

endclass

endpackage
```

5. Env

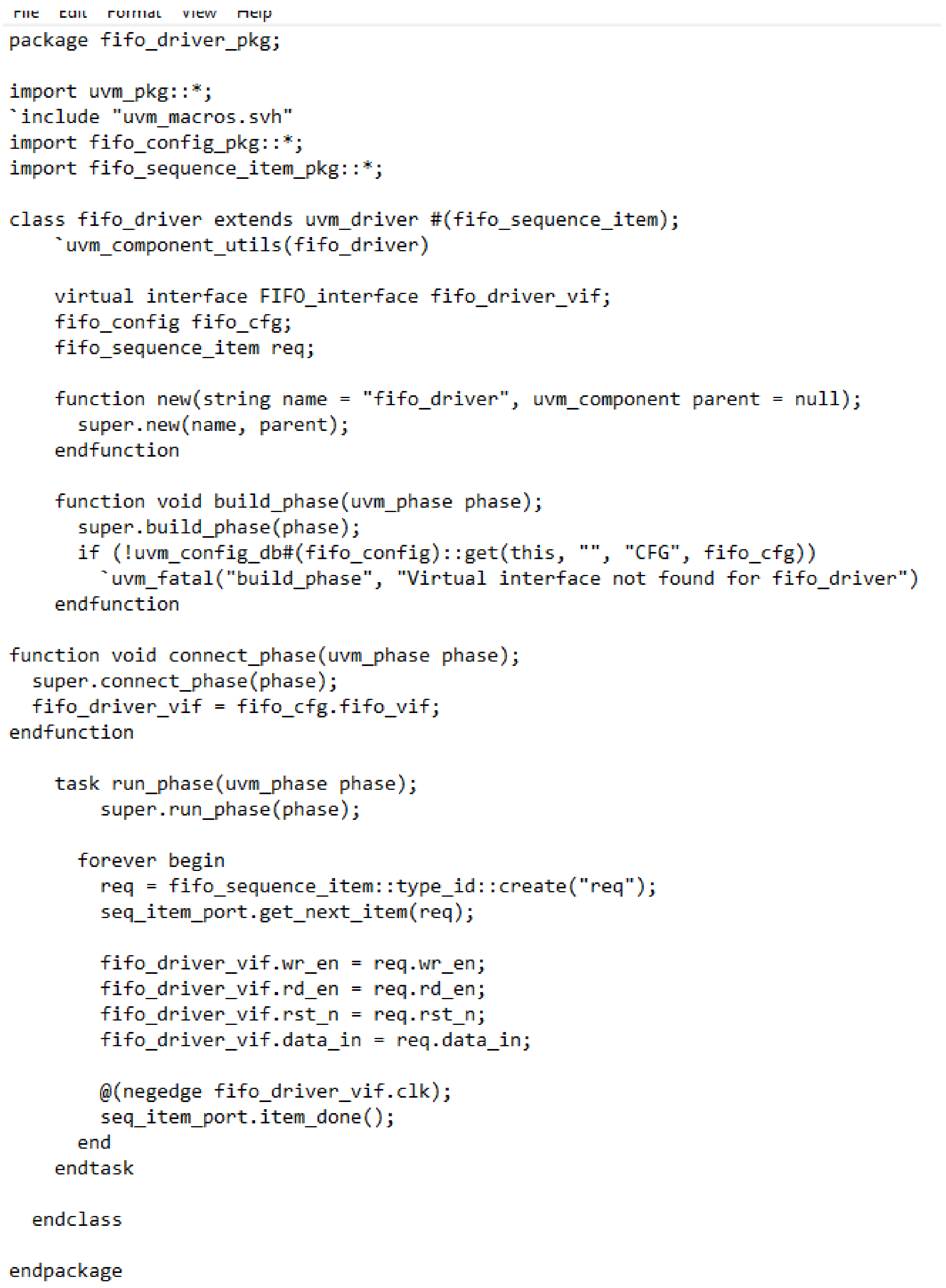
```
package fifo_env_pkg;

import uvm_pkg::*;
`include "uvm_macros.svh"
import fifo_agent_pkg::*;
import fifo_coverage_pkg::*;
import fifo_scoreboard_pkg::*;

  class fifo_env extends uvm_env;
    `uvm_component_utils(fifo_env)

    fifo_agent         m_agent;
    fifo_scoreboard    m_scoreboard;
    fifo_coverage      m_coverage;

    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);
      m_agent      = fifo_agent::type_id::create("m_agent", this);
      m_coverage   = fifo_coverage::type_id::create("m_coverage", this);
      m_scoreboard = fifo_scoreboard::type_id::create("m_scoreboard", this);

    endfunction

    function void connect_phase(uvm_phase phase);

      m_agent.agt_ap.connect(m_scoreboard.sb_export);
      m_agent.agt_ap.connect(m_coverage.cov_ap);

    endfunction

endclass

endpackage
```

6. Agent



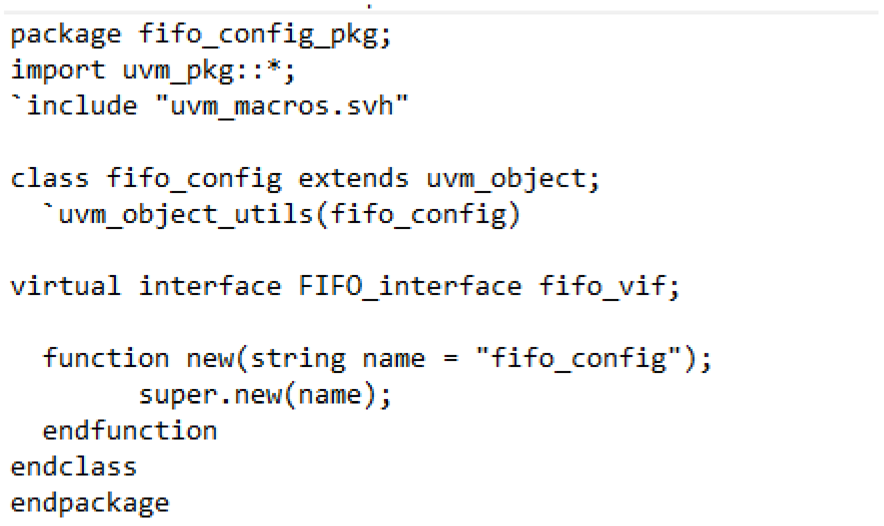
```
package fifo_agent_pkg;

  import uvm_pkg::*;
  `include "uvm_macros.svh"
  import fifo_config_pkg::*;
  import fifo_sequence_item_pkg::*;
  import fifo_sequencer_pkg::*;
  import fifo_driver_pkg::*;
  import fifo_monitor_pkg::*;

  class fifo_agent extends uvm_agent;
    `uvm_component_utils(fifo_agent)

    fifo_driver    m_driver;
    fifo_monitor   m_monitor;
    fifo_sequencer m_sequencer;
    fifo_config fifo_cfg;
    uvm_analysis_port #(fifo_sequence_item) agt_ap;

    function new(string name, uvm_component parent);
      super.new(name, parent);
    endfunction

    function void build_phase(uvm_phase phase);
      super.build_phase(phase);

      if (!uvm_config_db#(fifo_config)::get(this, "", "CFG", fifo_cfg))
        `uvm_fatal("AGENT_VIF", "Failed to get fifo_vif")

      m_driver    = fifo_driver::type_id::create("m_driver", this);
      m_monitor   = fifo_monitor::type_id::create("m_monitor", this);
      m_sequencer = fifo_sequencer::type_id::create("m_sequencer", this);
      agt_ap = new("agt_ap", this);
    endfunction

    function void connect_phase(uvm_phase phase);
        super.connect_phase(phase);
        m_driver.fifo_driver_vif = fifo_cfg.fifo_vif;
        m_monitor.fifo_mon_vif = fifo_cfg.fifo_vif;
        m_driver.seq_item_port.connect(m_sequencer.seq_item_export);
        m_monitor.mon_ap.connect(agt_ap);
    endfunction
  endclass

endpackage
```

7. Driver



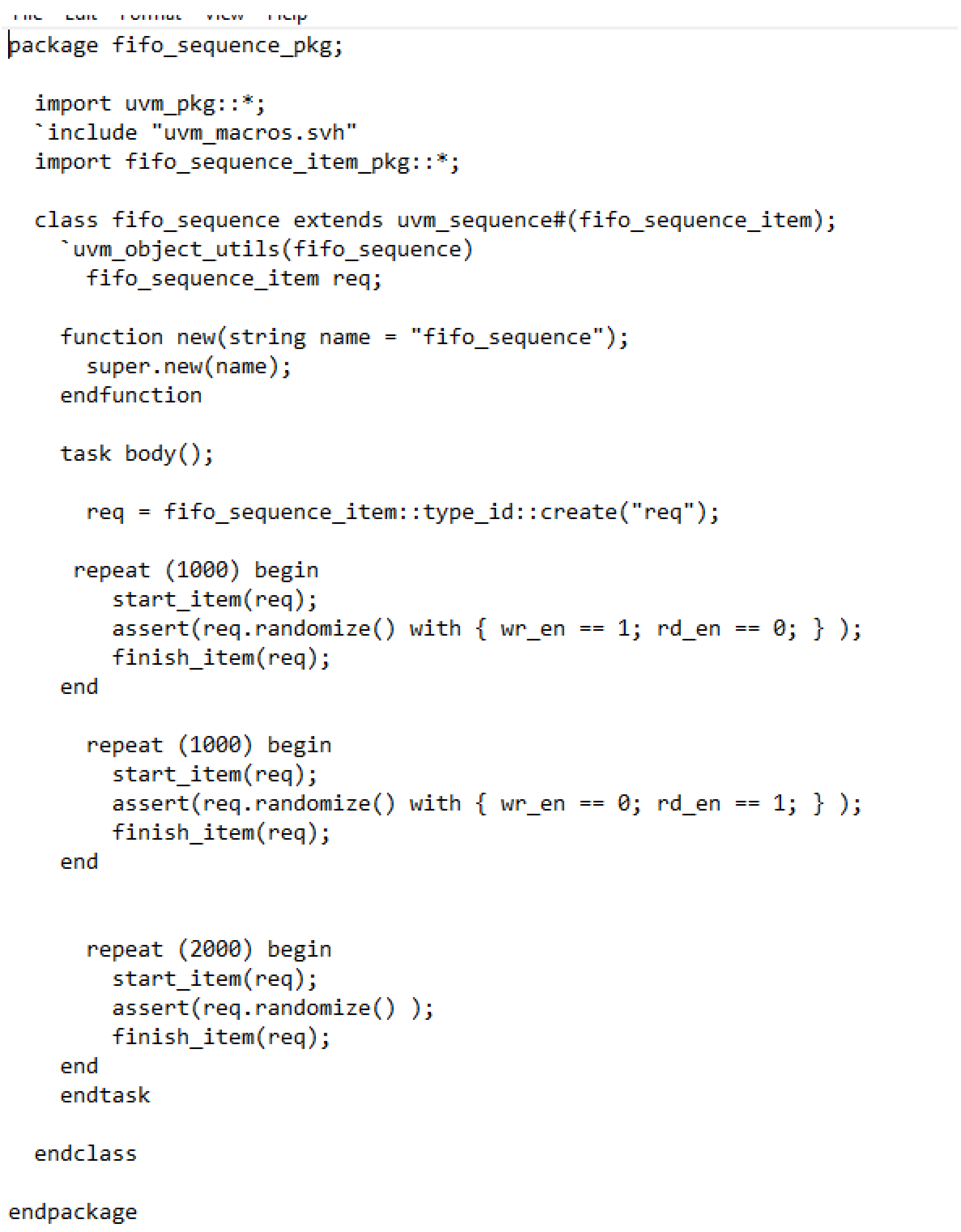
```
package fifo_driver_pkg;

import uvm_pkg::*;
`include "uvm_macros.svh"
import fifo_config_pkg::*;
import fifo_sequence_item_pkg::*;

class fifo_driver extends uvm_driver #(fifo_sequence_item);
    `uvm_component_utils(fifo_driver)

    virtual interface FIFO_interface fifo_driver_vif;
    fifo_config fifo_cfg;
    fifo_sequence_item req;

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

    function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      if (!uvm_config_db#(fifo_config)::get(this, "", "CFG", fifo_cfg))
        `uvm_fatal("build_phase", "Virtual interface not found for fifo_driver")
    endfunction

function void connect_phase(uvm_phase phase);
  super.connect_phase(phase);
  fifo_driver_vif = fifo_cfg.fifo_vif;
endfunction

    task run_phase(uvm_phase phase);
        super.run_phase(phase);

      forever begin
        req = fifo_sequence_item::type_id::create("req");
        seq_item_port.get_next_item(req);

        fifo_driver_vif.wr_en = req.wr_en;
        fifo_driver_vif.rd_en = req.rd_en;
        fifo_driver_vif.rst_n = req.rst_n;
        fifo_driver_vif.data_in = req.data_in;

        @(negedge fifo_driver_vif.clk);
        seq_item_port.item_done();
      end
    endtask

  endclass

endpackage
```

8. Config



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

class fifo_config extends uvm_object;
  `uvm_object_utils(fifo_config)

virtual interface FIFO_interface fifo_vif;

  function new(string name = "fifo_config");
        super.new(name);
  endfunction
endclass
endpackage
```

9. Sequence

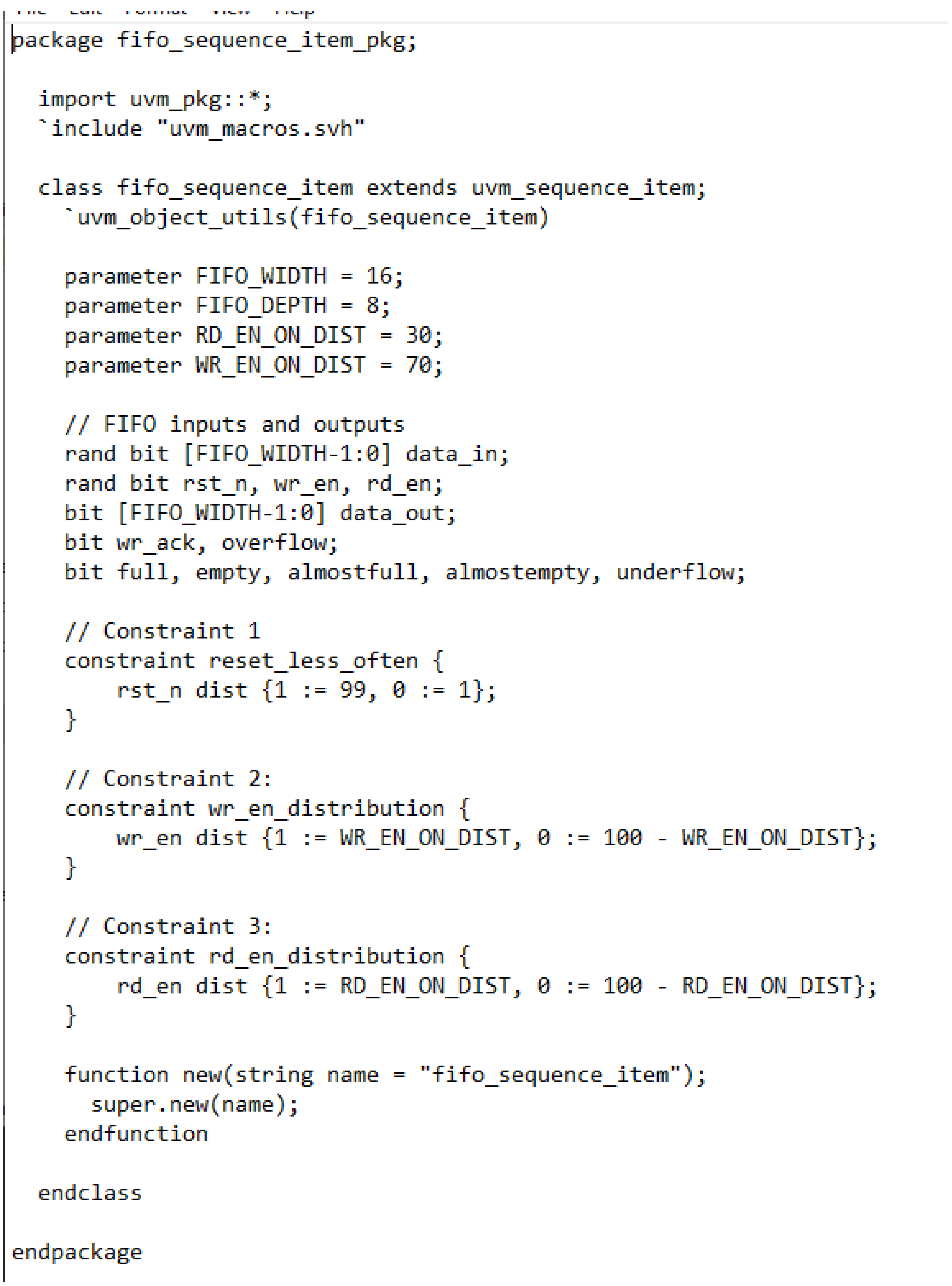


```
package fifo_sequence_pkg;

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

  class fifo_sequence extends uvm_sequence#(fifo_sequence_item);
    `uvm_object_utils(fifo_sequence)
      fifo_sequence_item req;

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

    task body();

      req = fifo_sequence_item::type_id::create("req");

     repeat (1000) begin
        start_item(req);
        assert(req.randomize() with { wr_en == 1; rd_en == 0; } );
        finish_item(req);
    end

      repeat (1000) begin
        start_item(req);
        assert(req.randomize() with { wr_en == 0; rd_en == 1; } );
        finish_item(req);
    end


      repeat (2000) begin
        start_item(req);
        assert(req.randomize() );
        finish_item(req);
    end
    endtask

  endclass

endpackage
```

10. Reset Sequence

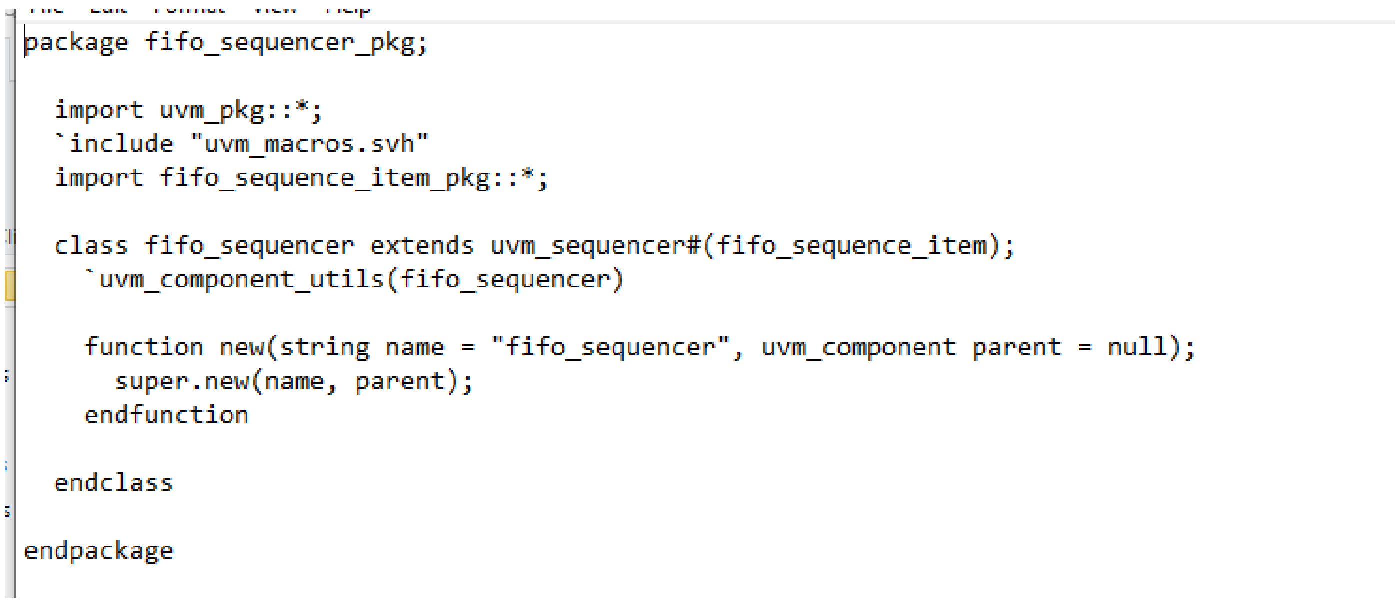
```
package fifo_reset_sequence_pkg;

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

  class fifo_reset_sequence extends uvm_sequence#(fifo_sequence_item);
    `uvm_object_utils(fifo_reset_sequence)
      fifo_sequence_item req;

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

    task body();

      req = fifo_sequence_item::type_id::create("req");
        start_item(req);
        req.rst_n=0;
        finish_item(req);

    endtask

  endclass

endpackage
```

11. Sequence Item



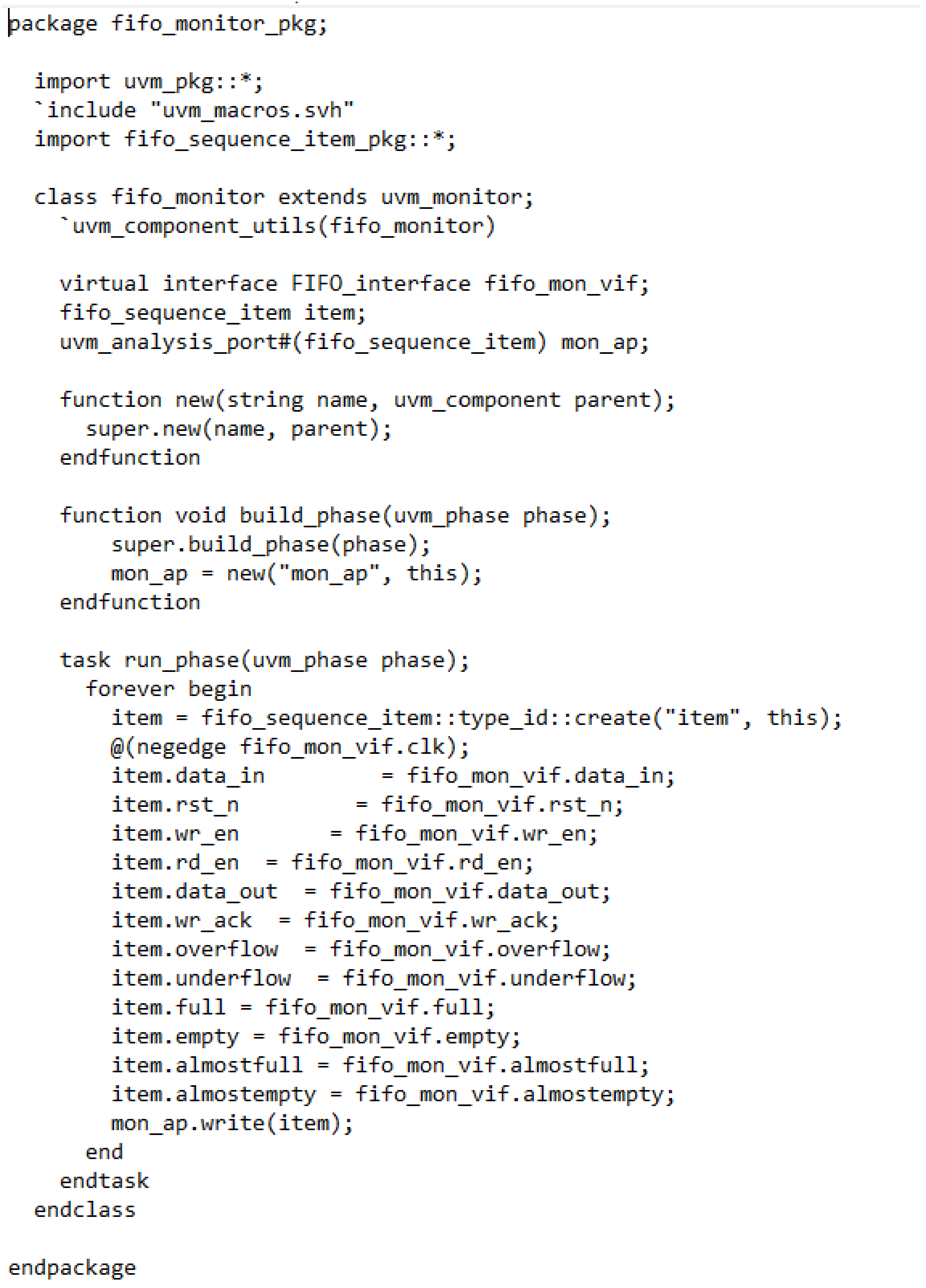
```
package fifo_sequence_item_pkg;

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

  class fifo_sequence_item extends uvm_sequence_item;
    `uvm_object_utils(fifo_sequence_item)

    parameter FIFO_WIDTH = 16;
    parameter FIFO_DEPTH = 8;
    parameter RD_EN_ON_DIST = 30;
    parameter WR_EN_ON_DIST = 70;

    // FIFO inputs and outputs
    rand bit [FIFO_WIDTH-1:0] data_in;
    rand bit rst_n, wr_en, rd_en;
    bit [FIFO_WIDTH-1:0] data_out;
    bit wr_ack, overflow;
    bit full, empty, almostfull, almostempty, underflow;

    // Constraint 1
    constraint reset_less_often {
        rst_n dist {1 := 99, 0 := 1};
    }

    // Constraint 2:
    constraint wr_en_distribution {
        wr_en dist {1 := WR_EN_ON_DIST, 0 := 100 - WR_EN_ON_DIST};
    }

    // Constraint 3:
    constraint rd_en_distribution {
        rd_en dist {1 := RD_EN_ON_DIST, 0 := 100 - RD_EN_ON_DIST};
    }

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

  endclass

endpackage
```

12. Sequencer



```
package fifo_sequencer_pkg;

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

  class fifo_sequencer extends uvm_sequencer#(fifo_sequence_item);
    `uvm_component_utils(fifo_sequencer)

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

  endclass

endpackage
```

13. Monitor



```
package fifo_monitor_pkg;

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

  class fifo_monitor extends uvm_monitor;
    `uvm_component_utils(fifo_monitor)

    virtual interface FIFO_interface fifo_mon_vif;
    fifo_sequence_item item;
    uvm_analysis_port#(fifo_sequence_item) mon_ap;

    function new(string name, uvm_component parent);
      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);
      forever begin
        item = fifo_sequence_item::type_id::create("item", this);
        @(negedge fifo_mon_vif.clk);
        item.data_in         = fifo_mon_vif.data_in;
        item.rst_n         = fifo_mon_vif.rst_n;
        item.wr_en       = fifo_mon_vif.wr_en;
        item.rd_en  = fifo_mon_vif.rd_en;
        item.data_out  = fifo_mon_vif.data_out;
        item.wr_ack  = fifo_mon_vif.wr_ack;
        item.overflow  = fifo_mon_vif.overflow;
        item.underflow  = fifo_mon_vif.underflow;
        item.full = fifo_mon_vif.full;
        item.empty = fifo_mon_vif.empty;
        item.almostfull = fifo_mon_vif.almostfull;
        item.almostempty = fifo_mon_vif.almostempty;
        mon_ap.write(item);
      end
    endtask
  endclass

endpackage
```

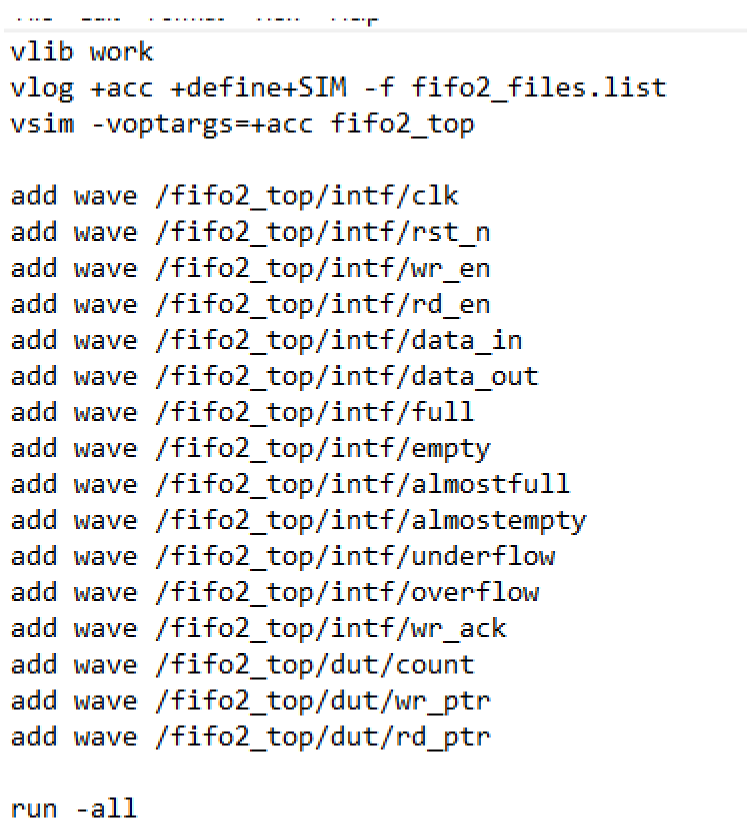
14. Coverage Collecter

```
package fifo_coverage_pkg;

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

class fifo_coverage extends uvm_component;
  `uvm_component_utils(fifo_coverage)

  uvm_analysis_export #(fifo_sequence_item) cov_ap;
  uvm_tlm_analysis_fifo #(fifo_sequence_item) cov_fifo;
  fifo_sequence_item item;

  // Mirror interface signals here
    logic wr_en, rd_en;

    logic wr_ack, overflow;
    logic full, empty, almostfull, almostempty, underflow;


  covergroup CVG;

    // Coverpoints
    wr_en_cp:           coverpoint wr_en { bins active = {1}; bins inactive = {0}; }
    rd_en_cp:           coverpoint rd_en { bins active = {1}; bins inactive = {0}; }

    wr_ack_cp:          coverpoint wr_ack { bins active = {1}; bins inactive = {0}; }
    full_cp:            coverpoint full { bins active = {1}; bins inactive = {0}; }
    empty_cp:           coverpoint empty { bins active = {1}; bins inactive = {0}; }
    almostfull_cp:      coverpoint almostfull { bins active = {1}; bins inactive = {0}; }
    almostempty_cp:     coverpoint almostempty { bins active = {1}; bins inactive = {0};
    underflow_cp:       coverpoint underflow {bins active = {1};bins inactive = {0}; }
    overflow_cp:        coverpoint overflow {bins active = {1};bins inactive = {0}; }

    // 7 required cross coverages
    cross_wr_rd_wrack:        cross wr_en_cp, rd_en_cp, wr_ack_cp;
    cross_wr_rd_full:         cross wr_en_cp, rd_en_cp, full_cp;
    cross_wr_rd_empty:        cross wr_en_cp, rd_en_cp, empty_cp;
    cross_wr_rd_almostfull:   cross wr_en_cp, rd_en_cp, almostfull_cp;
    cross_wr_rd_almostempty:  cross wr_en_cp, rd_en_cp, almostempty_cp;
    cross_wr_rd_underflow:   cross wr_en_cp, rd_en_cp, underflow_cp;
    cross_wr_rd_overflow:  cross wr_en_cp, rd_en_cp, overflow_cp;

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

  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    cov_ap = new("cov_ap", this);
    cov_fifo = new("cov_fifo", this);

  endfunction

  function void connect_phase(uvm_phase phase);
    super.connect_phase(phase);
    cov_ap.connect(cov_fifo.analysis_export);
  endfunction

  task run_phase(uvm_phase phase);
    super.run_phase(phase);
    forever begin
      cov_fifo.get(item);

      wr_en          = item.wr_en;
      rd_en          = item.rd_en;
      wr_ack         = item.wr_ack;
      overflow       = item.overflow;
      full           = item.full;
      empty          = item.empty;
      almostfull     = item.almostfull;
      almostempty    = item.almostempty;
      underflow      = item.underflow;

      CVG.sample();

    end
  endtask

endclass

endpackage
```

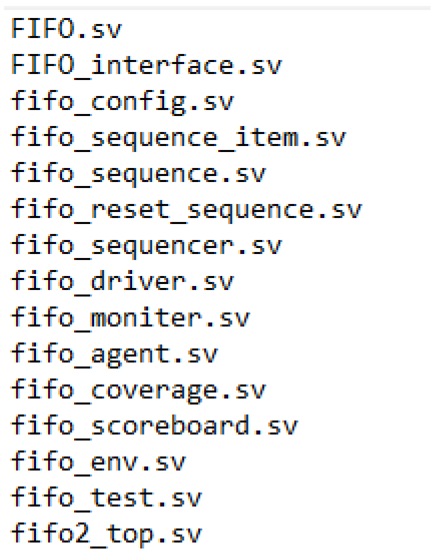
15. Scoreboard

```
package fifo_scoreboard_pkg;

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

  class fifo_scoreboard extends uvm_scoreboard;
    `uvm_component_utils(fifo_scoreboard)
    parameter FIFO_WIDTH = 16;
    parameter FIFO_DEPTH = 8;

    uvm_analysis_export #(fifo_sequence_item) sb_export;
    uvm_tlm_analysis_fifo #(fifo_sequence_item) sb_fifo;

    fifo_sequence_item seq_item_sb;

    int correct_count = 0;
    int error_count = 0;

    // Output reference variables
    bit [FIFO_WIDTH-1:0] data_out_ref;
    bit wr_ack_ref, overflow_ref;
    bit full_ref, empty_ref, almostfull_ref;
    bit almostempty_ref, underflow_ref;

    // Reference model
    function void reference_model(
      input bit [FIFO_WIDTH-1:0] data_out1,
      input bit wr_ack1, overflow1, full1, empty1,
      input bit almostfull1, almostempty1, underflow1
    );
      data_out_ref     = data_out1;
      wr_ack_ref       = wr_ack1;
      overflow_ref     = overflow1;
      full_ref         = full1;
      empty_ref        = empty1;
      almostfull_ref   = almostfull1;
      almostempty_ref  = almostempty1;
      underflow_ref    = underflow1;
    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(seq_item_sb);
        reference_model(seq_item_sb.data_out, seq_item_sb.wr_ack, seq_item_sb.overflow,
                        seq_item_sb.full, seq_item_sb.empty, seq_item_sb.almostfull,
                        seq_item_sb.almostempty, seq_item_sb.underflow);

        if (seq_item_sb.data_out != data_out_ref) begin
          `uvm_error("run_phase", $sformatf("Mismatch: DUT=%0d, REF=%0d",
                                            seq_item_sb.data_out, data_out_ref))
          error_count++;
        end
        else begin
          correct_count++;
        end
      end
    endtask

    function void report_phase(uvm_phase phase);
      super.report_phase(phase);
      `uvm_info("report_phase", $sformatf("Correct transactions: %0d", correct_count), UVM
      `uvm_info("report_phase", $sformatf("Failed transactions: %0d", error_count), UVM_ME
    endfunction


  endclass

  endpackage
```

16. Do file & alsu_files

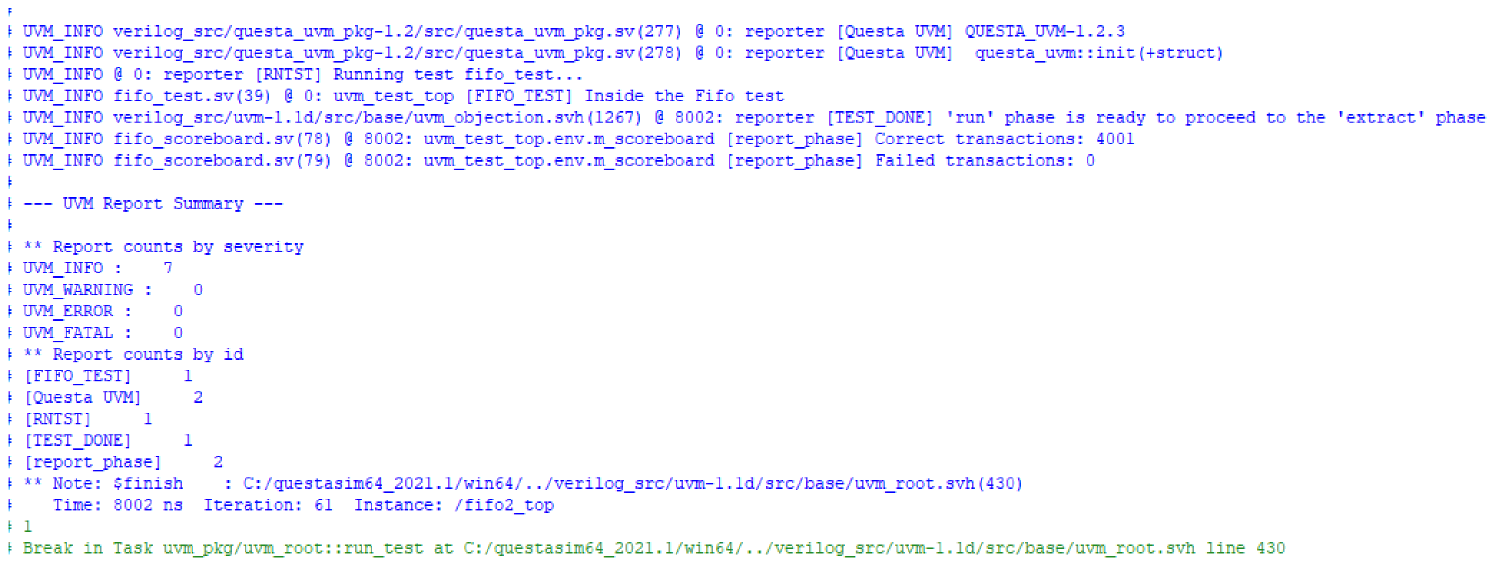


```
vlib work
vlog +acc +define+SIM -f fifo2_files.list
vsim -voptargs=+acc fifo2_top

add wave /fifo2_top/intf/clk
add wave /fifo2_top/intf/rst_n
add wave /fifo2_top/intf/wr_en
add wave /fifo2_top/intf/rd_en
add wave /fifo2_top/intf/data_in
add wave /fifo2_top/intf/data_out
add wave /fifo2_top/intf/full
add wave /fifo2_top/intf/empty
add wave /fifo2_top/intf/almostfull
add wave /fifo2_top/intf/almostempty
add wave /fifo2_top/intf/underflow
add wave /fifo2_top/intf/overflow
add wave /fifo2_top/intf/wr_ack
add wave /fifo2_top/dut/count
add wave /fifo2_top/dut/wr_ptr
add wave /fifo2_top/dut/rd_ptr

run -all
```



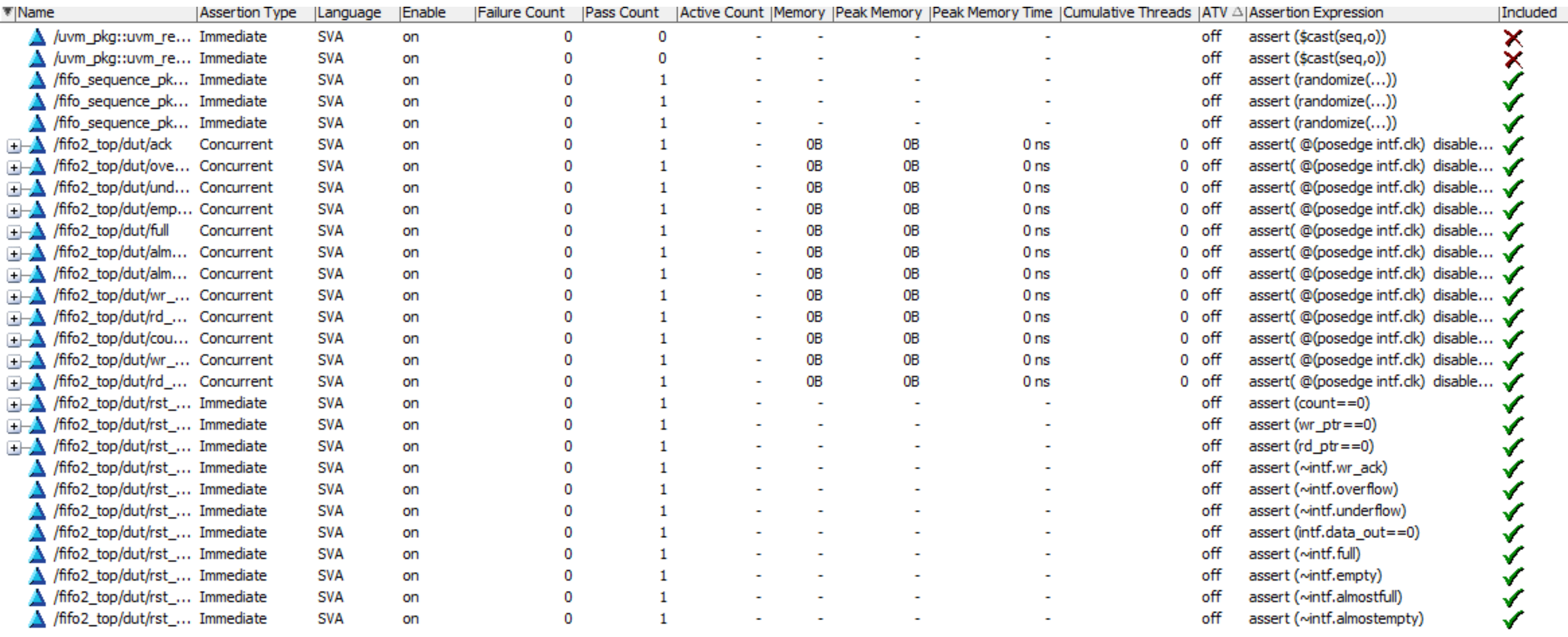
```
FIFO.sv
FIFO_interface.sv
fifo_config.sv
fifo_sequence_item.sv
fifo_sequence.sv
fifo_reset_sequence.sv
fifo_sequencer.sv
fifo_driver.sv
fifo_moniter.sv
fifo_agent.sv
fifo_coverage.sv
fifo_scoreboard.sv
fifo_env.sv
fifo_test.sv
fifo2_top.sv
```

17. Transcript

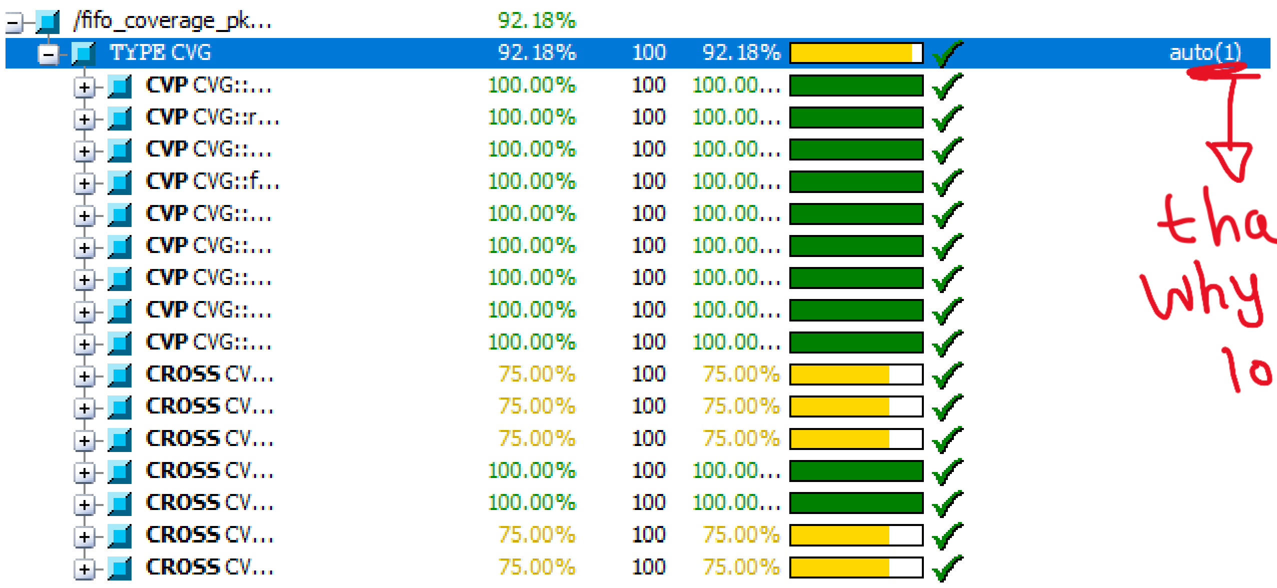


```
#
# UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(277) @ 0: reporter [Questa UVM] QUESTA_UVM-1.2.3
# UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(278) @ 0: reporter [Questa UVM]  questa_uvm::init(+struct)
# UVM_INFO @ 0: reporter [RNTST] Running test fifo_test...
# UVM_INFO fifo_test.sv(39) @ 0: uvm_test_top [FIFO_TEST] Inside the Fifo test
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 8002: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO fifo_scoreboard.sv(78) @ 8002: uvm_test_top.env.m_scoreboard [report_phase] Correct transactions: 4001
# UVM_INFO fifo_scoreboard.sv(79) @ 8002: uvm_test_top.env.m_scoreboard [report_phase] Failed transactions: 0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :    7
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [FIFO_TEST]     1
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# [report_phase]     2
# ** Note: $finish    : C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 8002 ns  Iteration: 61  Instance: /fifo2_top
# 1
# Break in Task uvm_pkg/uvm_root::run_test at C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh line 430
```

18. Assertions & Coverage



Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Count	Memory	Peak Memory	Peak Memory Time	Cumulative Threads	ATV	Assertion Expression	Included
/uvm_pkg::uvm_re…	Immediate	SVA	on	0	0	-	-	-	-		off	assert ($cast(seq,o))	✗
/uvm_pkg::uvm_re…	Immediate	SVA	on	0	0	-	-	-	-		off	assert ($cast(seq,o))	✗
/fifo_sequence_pk…	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/fifo_sequence_pk…	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/fifo_sequence_pk…	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/fifo2_top/dut/ack	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge intf.clk) disable…	✓
/fifo2_top/dut/ove…	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge intf.clk) disable…	✓
/fifo2_top/dut/und…	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge intf.clk) disable…	✓
/fifo2_top/dut/emp…	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge intf.clk) disable…	✓
/fifo2_top/dut/full	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge intf.clk) disable…	✓
/fifo2_top/dut/alm…	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge intf.clk) disable…	✓
/fifo2_top/dut/alm…	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge intf.clk) disable…	✓
/fifo2_top/dut/wr_…	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge intf.clk) disable…	✓
/fifo2_top/dut/rd_…	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge intf.clk) disable…	✓
/fifo2_top/dut/cou…	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge intf.clk) disable…	✓
/fifo2_top/dut/wr_…	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge intf.clk) disable…	✓
/fifo2_top/dut/rd_…	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge intf.clk) disable…	✓
/fifo2_top/dut/rst_…	Immediate	SVA	on	0	1	-	-	-	-		off	assert (count==0)	✓
/fifo2_top/dut/rst_…	Immediate	SVA	on	0	1	-	-	-	-		off	assert (wr_ptr==0)	✓
/fifo2_top/dut/rst_…	Immediate	SVA	on	0	1	-	-	-	-		off	assert (rd_ptr==0)	✓
/fifo2_top/dut/rst_…	Immediate	SVA	on	0	1	-	-	-	-		off	assert (~intf.wr_ack)	✓
/fifo2_top/dut/rst_…	Immediate	SVA	on	0	1	-	-	-	-		off	assert (~intf.overflow)	✓
/fifo2_top/dut/rst_…	Immediate	SVA	on	0	1	-	-	-	-		off	assert (~intf.underflow)	✓
/fifo2_top/dut/rst_…	Immediate	SVA	on	0	1	-	-	-	-		off	assert (intf.data_out==0)	✓
/fifo2_top/dut/rst_…	Immediate	SVA	on	0	1	-	-	-	-		off	assert (~intf.full)	✓
/fifo2_top/dut/rst_…	Immediate	SVA	on	0	1	-	-	-	-		off	assert (~intf.empty)	✓
/fifo2_top/dut/rst_…	Immediate	SVA	on	0	1	-	-	-	-		off	assert (~intf.almostfull)	✓
/fifo2_top/dut/rst_…	Immediate	SVA	on	0	1	-	-	-	-		off	assert (~intf.almostempty)	✓

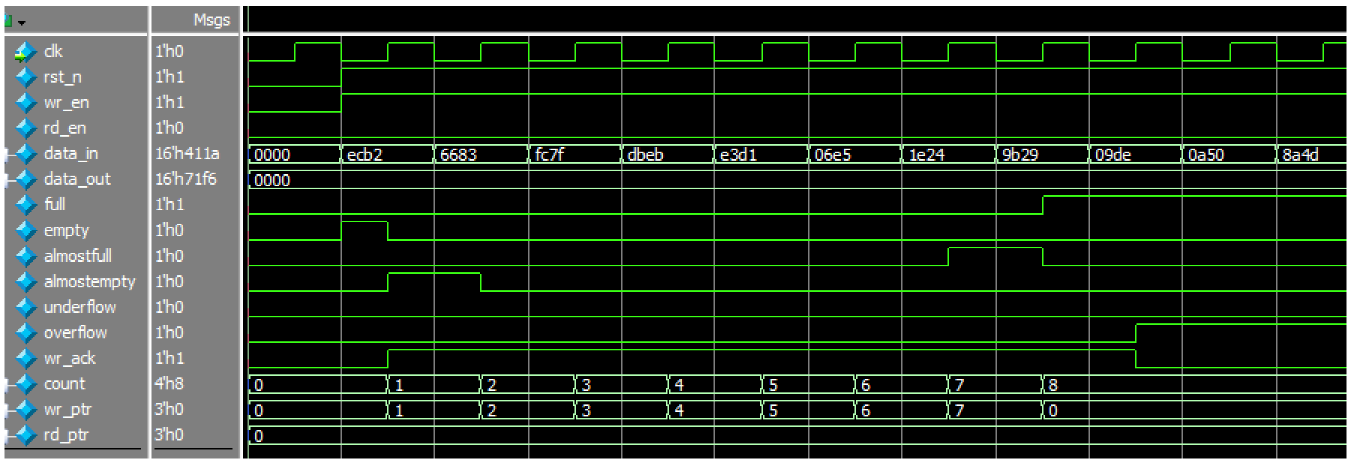


/fifo_coverage_pk…	92.18%					
TYPE CVG	92.18%	100	92.18%	🟨	✓	auto(1)
CVP CVG::…	100.00%	100	100.00…	🟩	✓	
CVP CVG::r…	100.00%	100	100.00…	🟩	✓	
CVP CVG::…	100.00%	100	100.00…	🟩	✓	
CVP CVG::f…	100.00%	100	100.00…	🟩	✓	
CVP CVG::…	100.00%	100	100.00…	🟩	✓	
CVP CVG::…	100.00%	100	100.00…	🟩	✓	
CVP CVG::…	100.00%	100	100.00…	🟩	✓	
CVP CVG::…	100.00%	100	100.00…	🟩	✓	
CVP CVG::…	100.00%	100	100.00…	🟩	✓	
CROSS CV…	75.00%	100	75.00%	🟨	✓	
CROSS CV…	75.00%	100	75.00%	🟨	✓	
CROSS CV…	75.00%	100	75.00%	🟨	✓	
CROSS CV…	100.00%	100	100.00…	🟩	✓	
CROSS CV…	100.00%	100	100.00…	🟩	✓	
CROSS CV…	75.00%	100	75.00%	🟨	✓	
CROSS CV…	75.00%	100	75.00%	🟨	✓	

$\downarrow$ thats why not 100%

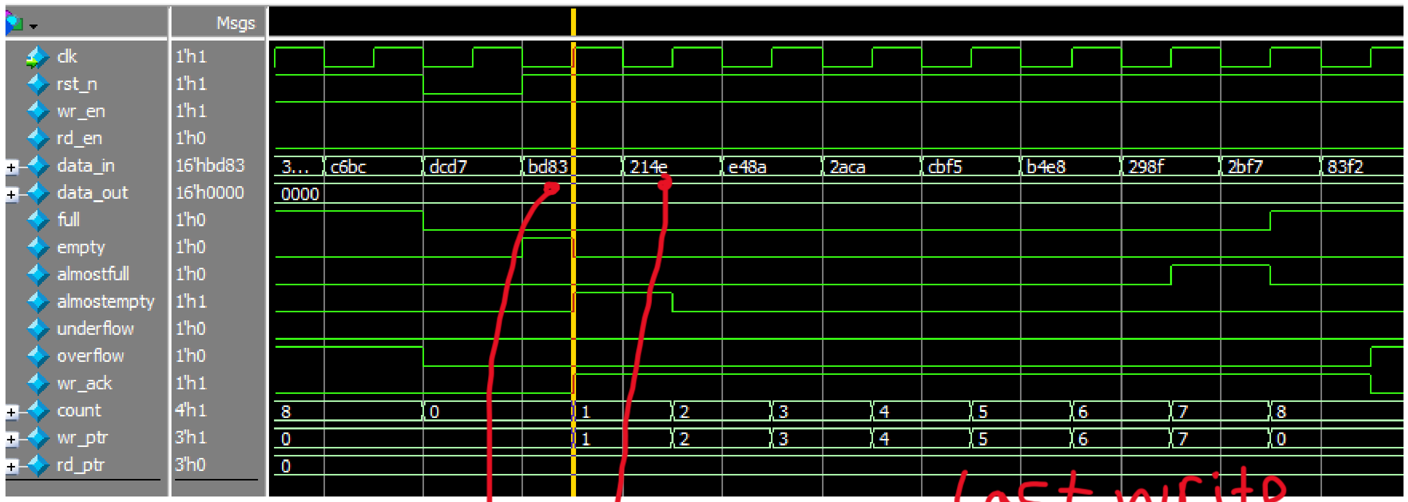
18. Waveform

Write only Sequence)



	Msgs
clk	1'h0
rst_n	1'h1
wr_en	1'h1
rd_en	1'h0
data_in	16'h411a
data_out	16'h71f6
full	1'h1
empty	1'h0
almostfull	1'h0
almostempty	1'h0
underflow	1'h0
overflow	1'h0
wr_ack	1'h1
count	4'h8
wr_ptr	3'h0
rd_ptr	3'h0

data_in: 0000 ecb2 6683 fc7f dbeb e3d1 06e5 1e24 9b29 09de 0a50 8a4d
data_out: 0000
count: 0 1 2 3 4 5 6 7 8
wr_ptr: 0 1 2 3 4 5 6 7 0
rd_ptr: 0

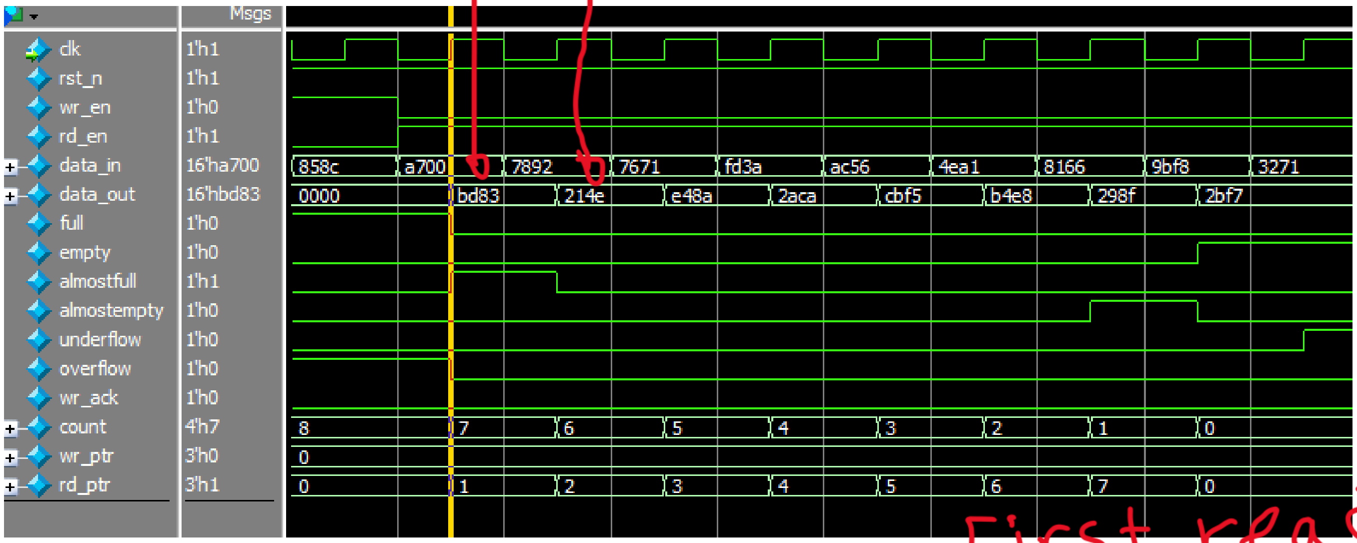


	Msgs
clk	1'h1
rst_n	1'h1
wr_en	1'h1
rd_en	1'h0
data_in	16'hbd83
data_out	16'h0000
full	1'h0
empty	1'h0
almostfull	1'h0
almostempty	1'h1
underflow	1'h0
overflow	1'h0
wr_ack	1'h1
count	4'h1
wr_ptr	3'h1
rd_ptr	3'h0

data_in: 3... c6bc dcd7 bd83 214e e48a 2aca cbf5 b4e8 298f 2bf7 83f2
data_out: 0000
count: 8 0 1 2 3 4 5 6 7 8
wr_ptr: 0 1 2 3 4 5 6 7 0
rd_ptr: 0

Last write

Read only Sequence)



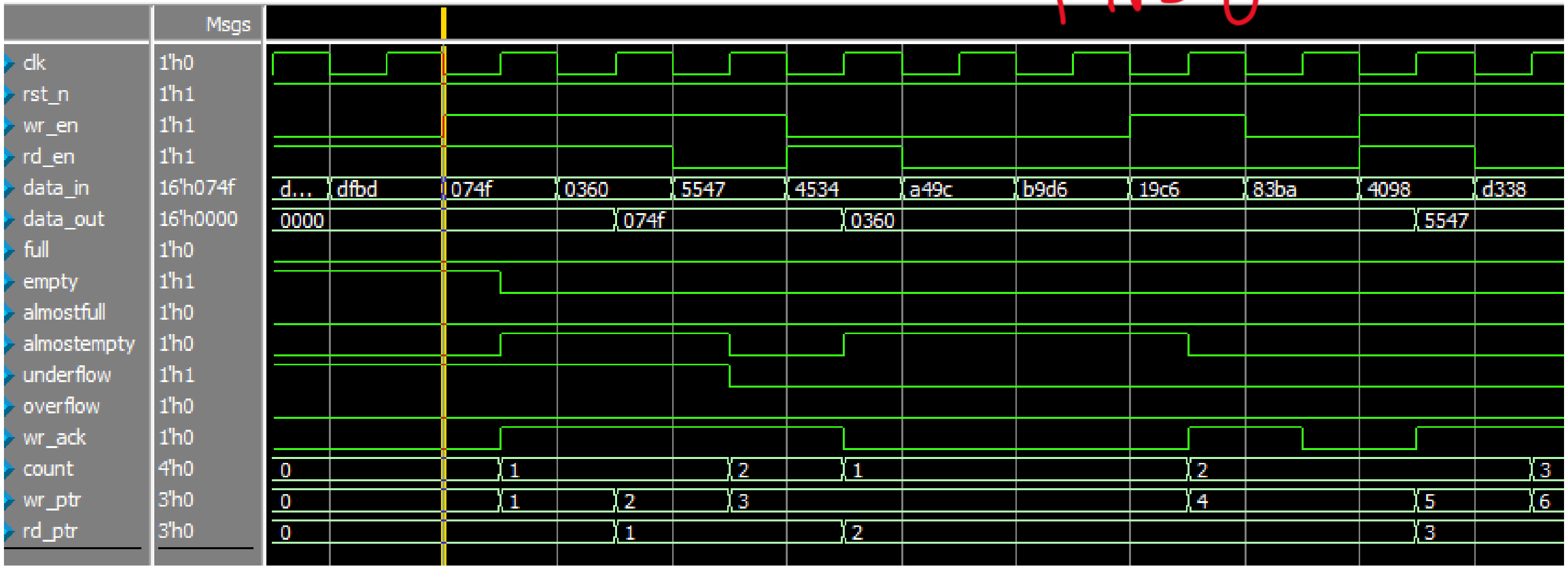
	Msgs
clk	1'h1
rst_n	1'h1
wr_en	1'h0
rd_en	1'h1
data_in	16'ha700
data_out	16'hbd83
full	1'h0
empty	1'h0
almostfull	1'h1
almostempty	1'h0
underflow	1'h0
overflow	1'h0
wr_ack	1'h0
count	4'h7
wr_ptr	3'h0
rd_ptr	3'h1

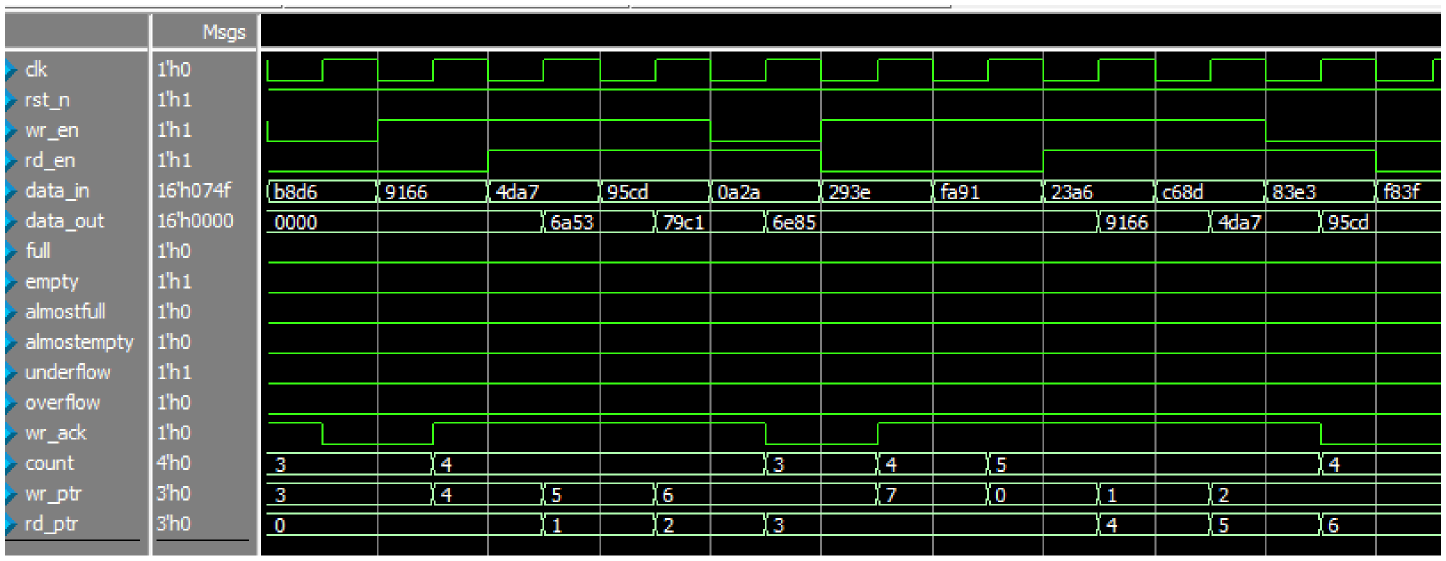
data_in: 858c a700 7892 7671 fd3a ac56 4ea1 8166 9bf8 3271
data_out: 0000 bd83 214e e48a 2aca cbf5 b4e8 298f 2bf7
count: 8 7 6 5 4 3 2 1 0
wr_ptr: 0
rd_ptr: 0 1 2 3 4 5 6 7 0

First read

Write_Read sequence)

First


Msgs
clk 1'h0
rst_n 1'h1
wr_en 1'h1
rd_en 1'h1
data_in 16'h074f
data_out 16'h0000
full 1'h0
empty 1'h1
almostfull 1'h0
almostempty 1'h0
underflow 1'h1
overflow 1'h0
wr_ack 1'h0
count 4'h0
wr_ptr 3'h0
rd_ptr 3'h0
d... dfbd 074f 0360 5547 4534 a49c b9d6 19c6 83ba 4098 d338
0000 074f 0360 5547
0 1 2 1 2 3
0 1 2 3 4 5 6
0 1 2 3


Msgs
clk 1'h0
rst_n 1'h1
wr_en 1'h1
rd_en 1'h1
data_in 16'h074f
data_out 16'h0000
full 1'h0
empty 1'h1
almostfull 1'h0
almostempty 1'h0
underflow 1'h1
overflow 1'h0
wr_ack 1'h0
count 4'h0
wr_ptr 3'h0
rd_ptr 3'h0
b8d6 9166 4da7 95cd 0a2a 293e fa91 23a6 c68d 83e3 f83f
0000 6a53 79c1 6e85 9166 4da7 95cd
3 4 3 4 5 4
3 4 5 6 7 0 1 2
0 1 2 3 4 5 6