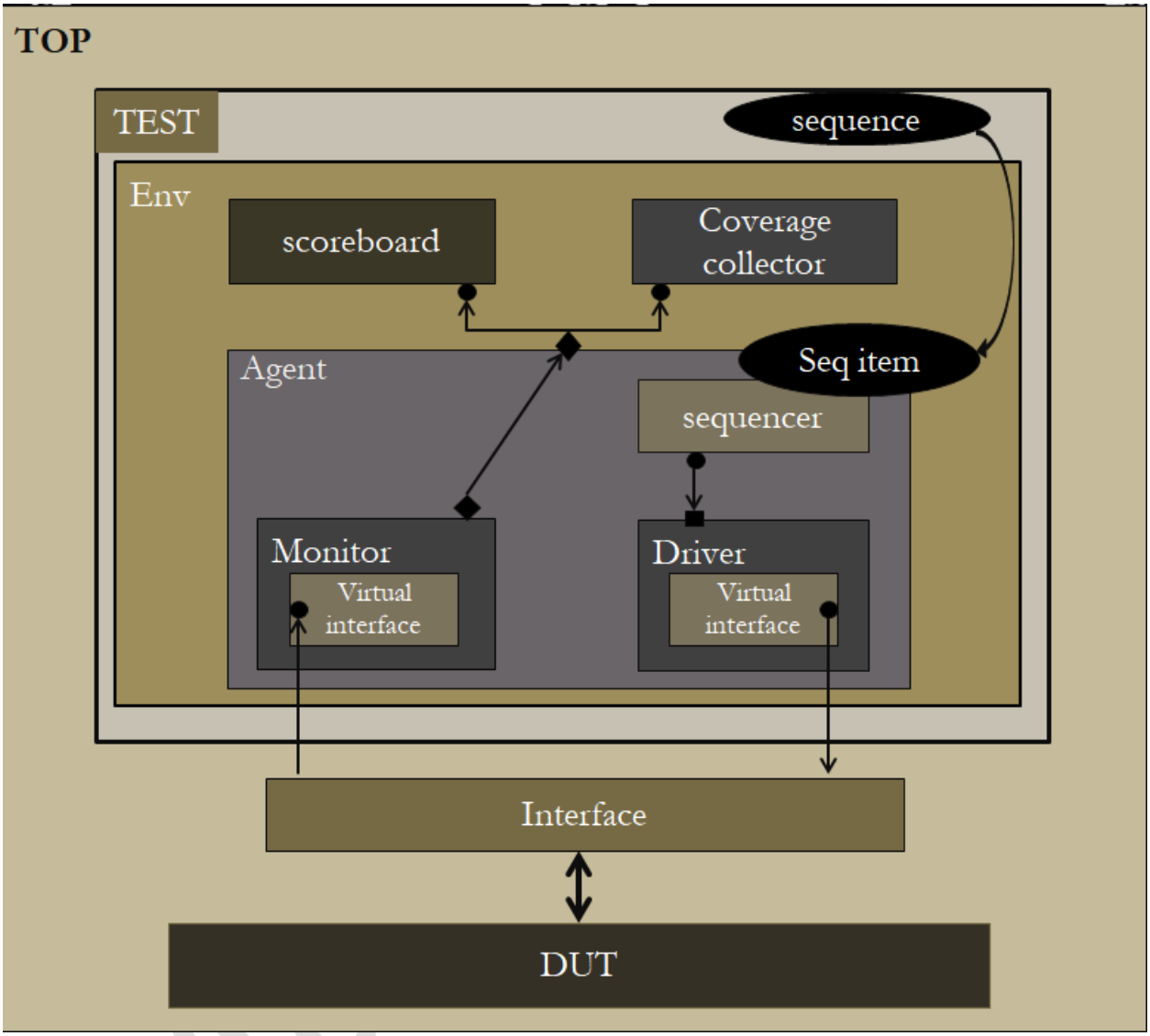
SYNCHRONOUS FIFO UVM VERIFICATION

Mark Amgad Saleh

Content:

UVM_Structure:



UVM_FLOW:

1. Top-Level Test Module (DUT + UVM Environment)

The top module instantiates both the **Design Under Test (DUT)** and the **UVM Environment**. It connects the DUT's physical signals to virtual interfaces, which are used by the UVM testbench to interact with the DUT.

- **DUT Instantiation**: The DUT is instantiated within the top module. All the necessary ports are exposed for connections.
- **Interface Definition**: A **SystemVerilog interface** is used to bundle the DUT's signals (input, output, control, clock, reset, etc.) into a single structure. This interface simplifies connections and enables the UVM testbench to drive and monitor the DUT signals.
- **Virtual Interface**: A `virtual interface` is passed to the UVM environment to enable interaction with the DUT. The virtual interface acts as a bridge between the physical signals of the DUT and the UVM components, allowing drivers and monitors to access and control signals without directly connecting to the DUT's ports.

2. UVM Environment

The **UVM Environment** is the top-level container for all verification components, such as agents, sequencers, drivers, monitors, and scoreboards. It coordinates the stimulus generation, driving, monitoring, and checking.

- **Agent**: The agent encapsulates all components needed to interact with the DUT (driver, sequencer, monitor). It can be active (driving stimulus) or passive (monitoring only).
  - **Active Agent**: Contains a driver and sequencer to generate and apply stimulus to the DUT.
  - **Passive Agent**: Contains a monitor that passively observes the DUT's responses without driving it.
- **Configuration**: The environment receives configuration objects that specify parameters (e.g., clock frequency, reset polarity) for use across all UVM components. The configuration is set in the UVM `build_phase`.

3. Driving the Interface (Driver and Sequencer)

The task of driving the DUT is performed by the **UVM Driver** and **UVM Sequencer**. The sequencer generates a series of transactions (test stimuli), which are then applied to the DUT through the driver.

- **Sequencer (Transaction Generator)**:
  - The sequencer generates a sequence of transactions (`uvm_sequence_item`), which represents the stimulus or data to be applied to the DUT.
  - It can generate both random and directed stimuli, depending on the needs of the test case. The sequence items can be randomized with constraints to explore different functional corners of the design.

    - The sequencer passes these transactions to the driver via the `uvm_driver-uvm_sequencer` handshake mechanism.
- **Driver (Signal Driver)**:
    - The driver receives the transaction from the sequencer and converts it into physical signal activity on the DUT interface.
    - The driver operates in cycles, waiting for new transactions from the sequencer, translating them into actual stimulus (signal toggles), and driving the DUT through the virtual interface.
    - The driver continuously monitors the state of the DUT (e.g., clock and ready signals) and synchronizes the stimulus application accordingly.

4. Monitoring the DUT (Monitor)

The **UVM Monitor** passively observes the signals on the DUT interface and captures the response for later checking and coverage collection. Unlike the driver, the monitor does not interact with the DUT directly (it does not drive signals); instead, it records the signal values as they change during the test.

- **Passive Observation**: The monitor is connected to the DUT signals through the same virtual interface as the driver. It continuously samples the DUT outputs and records the behavior.
- **Transaction Reconstruction**: The monitor collects data from the DUT and reconstructs the transactions as they appear at the output. This reconstructed transaction is then sent to other UVM components, such as the scoreboard.
- **Coverage Collection**: Functional coverage is often collected within the monitor. Coverage points and bins are defined to ensure that the test explores all intended functionalities. Monitors track the coverage of events, state transitions, or specific combinations of input/output values.

5. Analyzing the Output (Scoreboard and Checkers)

The **UVM Scoreboard** is responsible for comparing the DUT's actual output to the expected output. It determines whether the DUT behavior matches the design specification.

- **Expected Results Generation**: The scoreboard receives expected transaction data, which can be generated by:
    - A **reference model** (golden model) that simulates the ideal behavior of the DUT.
    - Direct comparison based on known inputs and expected outputs for directed tests.
- **Result Comparison**: The scoreboard compares the actual output captured by the monitor with the expected results. It flags any mismatches as errors.
    - If the actual transaction matches the expected transaction, the test continues.
    - If there is a mismatch, the scoreboard logs an error, providing detailed information about the discrepancy (e.g., values, cycle, location).
- **Self-Checking Mechanism**: The scoreboard can automatically check the correctness of each transaction. Any differences between expected and actual results trigger an error report, which can halt the test or continue depending on the severity.

6. Coverage and Reporting

In addition to functional checks, coverage metrics are collected throughout the test.

- **Functional Coverage**: Defined using covergroups, functional coverage tracks which features of the DUT were exercised during testing. These covergroups are placed in the monitor or scoreboard and track coverage of events like valid transactions, protocol violations, and corner cases.
- **Code Coverage**: Tool-based code coverage analyzes which lines, conditions, branches, and states in the RTL code were exercised by the test. This coverage data is merged and reported at the end of the test.
- **Report Phase**: After the simulation run completes, UVM generates a detailed report, summarizing errors, functional coverage, and code coverage. The user can analyze the coverage reports and logs to determine if the verification goals have been met.

Verification Plan:

Steps of verification:

- assert reset seq item.
- Write only sequence item and make the FIFO full.
- Read only sequence item and make the FIFO empty.
- apply write and read seq item .
- assert reset seq item .

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 with assertion and scoreboard check data function
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 with assertion and scoreboard check data function
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 with assertion and scoreboard check data function
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 rd_en and underflow	Output checked with assertion and scoreboard check data function
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 with assertion and scoreboard check data function

FIFO_6	In case of both read enable and write enable are HIGH and the FIFO is full, then only read occurs	Randomization on wr_en and rd_en under constraint that write occurs more than read	Included as cross cover for wr_en and rd_en and all the output flags	Output checked with assertion and scoreboard check data function
FIFO_7	In case of both read enable and write enable are HIGH and the FIFO is empty, then only write occurs	Randomization on wr_en and rd_en under constraint that write occurs more than read	Included as cross cover for wr_en and rd_en and all the output flags	Output checked with assertion and scoreboard check data function
FIFO_8	In case of both read enable and write enable are HIGH and the FIFO is not full or empty, then both write and read occur	Randomization on wr_en and rd_en under constraint that write occurs more than read	Included as cross cover for wr_en and rd_en and all the output flags	Output checked with assertion and scoreboard check data function
FIFO_9	data_out, at start is invalid due to initial state	Randomization on rst_n under constraint that reset is deasserted most of the time	Not included	Output checked with assertion and scoreboard check data function
FIFO_10	If the reset is asserted, we will be at the initial state	Randomization on rst_n under constraint that reset is deasserted most of the time	Not included	Output checked with assertion and scoreboard check data function

Bug Report:

FIFO after fixing the following Bugs:

1. bug detected : Reset should makes only these seq outputs(overflow, under flow, wr_ack).

2. bug detected : output (underflow), It must be sequential not combinational output.

3. bug detected : output(almostfull), It must = 1 ,if count = FIFO_DEPTH - 1 , not FIFO_DEPTH – 2.

4. bug detected : the third always block should contain case of (1,1) to ensure that note “”” If a read and write enables were high and the FIFO was empty, only writing will take place and vice verse if the FIFO was full.”””

Assertion table:

Feature	Assertion
Overflow is asserted when count == FIFO_DEPTH and wr_en is HIGH	`overflow_assert_1: assert property (@(posedge clk) disable iff(!rst_n) ((count == FIFO_DEPTH) && wr_en)
Underflow is asserted when count == 0 and rd_en is HIGH	`underflow_assert_1: assert property (@(posedge clk) disable iff(!rst_n) ((count == 0) && rd_en)
Write ack is asserted when count != FIFO_DEPTH and wr_en is HIGH	`wr_ack_assert_1: assert property (@(posedge clk) disable iff(!rst_n) ((count != FIFO_DEPTH) && wr_en)
Overflow is not asserted when count != FIFO_DEPTH and wr_en is HIGH	`overflow_assert_2: assert property (@(posedge clk) disable iff(!rst_n) ((count != FIFO_DEPTH) && wr_en)
Underflow is not asserted when count != 0 and rd_en is HIGH	`underflow_assert_2: assert property (@(posedge clk) disable iff(!rst_n) ((count != 0) && rd_en)
Write ack is not asserted when count == FIFO_DEPTH and wr_en is HIGH	`wr_ack_assert_2: assert property (@(posedge clk) disable iff(!rst_n) ((count == FIFO_DEPTH) && wr_en)
Full is asserted when count == FIFO_DEPTH	`full_assert_1: assert property (@(posedge clk) (count == FIFO_DEPTH)
Empty is asserted when count == 0	`empty_assert_1: assert property (@(posedge clk) (count == 0)
Almost full is asserted when count == FIFO_DEPTH - 1	`almostfull_assert_1: assert property (@(posedge clk) (count == FIFO_DEPTH-1)
Almost empty is asserted when count == 1	`almostempty_assert_1: assert property (@(posedge clk) (count == 1)
Full is not asserted when count != FIFO_DEPTH	`full_assert_2: assert property (@(posedge clk) (count != FIFO_DEPTH)
Empty is not asserted when count != 0	`empty_assert_2: assert property (@(posedge clk) (count != 0)
Almost full is not asserted when count != FIFO_DEPTH - 1	`almostfull_assert_2: assert property (@(posedge clk) (count != FIFO_DEPTH-1)
Almost empty is not asserted when count != 1	`almostempty_assert_2: assert property (@(posedge clk) (count != 1)
count decrements when rd_en is HIGH, wr_en is LOW, and count != 0	`rd_count_assert: assert property (@(posedge clk) disable iff(!rst_n) (rd_en && !wr_en && count != 0)
count increments when wr_en is HIGH, rd_en is LOW, and count != FIFO_DEPTH	`wr_count_assert: assert property (@(posedge clk) disable iff(!rst_n) (!rd_en && wr_en && count != FIFO_DEPTH)
count remains the same when both rd_en and wr_en are HIGH	`rd_wr_count_assert: assert property (@(posedge clk) disable iff(!rst_n) (rd_en && wr_en && count != 0 && count != FIFO_DEPTH)
Read pointer increments when rd_en is HIGH and count != 0	`rd_ptr_assert: assert property (@(posedge clk) disable iff(!rst_n) (rd_en && count != 0)
count remains the same when both rd_en and wr_en are HIGH	`rd_wr_count_assert: assert property (@(posedge clk) disable iff(!rst_n) (rd_en && wr_en && count != 0 && count != FIFO_DEPTH)
Read pointer increments when rd_en is HIGH and count != 0	`rd_ptr_assert: assert property (@(posedge clk) disable iff(!rst_n) (rd_en && count != 0)
Write pointer increments when wr_en is HIGH and count != FIFO_DEPTH	`wr_ptr_assert: assert property (@(posedge clk) disable iff(!rst_n) (wr_en && count != FIFO_DEPTH)
Reset assertions for count, pointers, and flags	always comb block

Shared package:

```
package shared_pkg;
    parameter FIFO_WIDTH    = 16;
    parameter FIFO_DEPTH    = 8;
    parameter max_fifo_addr = $clog2(FIFO_DEPTH);
endpackage
```

Design:

```
import shared_pkg::*;
module FIFO(clk, rst_n, wr_en, rd_en, data_in, data_out, wr_ack, overflow, full, empty, almostfull, almostempty, underflow);

    input [FIFO_WIDTH-1:0] data_in;
    input clk, rst_n, wr_en, rd_en;
    output reg [FIFO_WIDTH-1:0] data_out;
    output full, empty, almostfull, almostempty ;
    output reg overflow , underflow , wr_ack;

    reg [FIFO_WIDTH    -1 :0] mem [FIFO_DEPTH - 1:0];
    reg [max_fifo_addr -1 :0] wr_ptr , rd_ptr;
    reg [max_fifo_addr    :0] count;

    always @(posedge clk or negedge rst_n) begin //////write operation////////
        if (!rst_n) begin
            wr_ptr   <= 0;
            wr_ack   <= 0;
            overflow <= 0;
        end
        else if (wr_en && count < FIFO_DEPTH) begin
            mem[wr_ptr] <= data_in;
            wr_ack      <= 1;
            wr_ptr      <= wr_ptr + 1;
            overflow    <= 0;
        end
        else begin
            wr_ack <= 0;
            if (wr_en && full)
                overflow <= 1;
            else
                overflow <= 0;
            end
    end

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

            if (rd_en && empty)
                underflow <= 1;
            else
                underflow <= 0;

        end
    end

    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
```

```
            if (rd_en && empty)
                underflow <= 1;
            else
                underflow <= 0;

        end
    end

    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            count <= 0;
        end
        else begin
            if      ( ({wr_en, rd_en} == 2'b10) && !full)
                    count <= count + 1;

            else if ( ({wr_en, rd_en} == 2'b01) && !empty)
                    count <= count - 1;

            else if   ({wr_en, rd_en} == 2'b11) begin

                if (full)
                    count <= count - 1;
                else if (empty)
                    count <= count + 1;
                //else
                    //count <= count;

            end
        end
    end

    assign full        = (count == FIFO_DEPTH)   ? 1 : 0;
    assign empty       = (count == 0)            ? 1 : 0;
    assign almostfull  = (count == FIFO_DEPTH-1) ? 1 : 0;
    assign almostempty = (count == 1)            ? 1 : 0;

endmodule
```

Interface:

```
import shared_pkg::*;
interface  FIFO_interface(clk);
    input 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, full, empty, almostfull, almostempty, underflow;
endinterface
```

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_interface fifo_vif;

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

endclass
endpackage
```

Top:

```
import uvm_pkg::*;
`include "uvm_macros.svh"
import fifo_test_pkg::*;
import shared_pkg::*;
module top();
    bit clk;
    initial begin
        clk = 0;
        forever
            #1 clk =~clk;
    end

    FIFO_interface fifo_if(clk);

    FIFO DUT(
        fifo_if.clk,
        fifo_if.rst_n,
        fifo_if.wr_en,
        fifo_if.rd_en,
        fifo_if.data_in,
        fifo_if.data_out,
        fifo_if.wr_ack,
        fifo_if.overflow,
        fifo_if.full,
        fifo_if.empty,
        fifo_if.almostfull,
        fifo_if.almostempty,
        fifo_if.underflow);

    bind FIFO assertions SVA(
        fifo_if.clk,
        fifo_if.rst_n,
        fifo_if.wr_en,
        fifo_if.rd_en,
        fifo_if.data_in,
        fifo_if.data_out,
        fifo_if.wr_ack,
        fifo_if.overflow,
        fifo_if.full,
        fifo_if.empty,
        fifo_if.almostfull,
        fifo_if.almostempty,
        fifo_if.underflow,
        DUT.wr_ptr,
        DUT.rd_ptr,
        DUT.count);

    initial begin
        uvm_config_db#(virtual FIFO_interface)::set(null, "uvm_test_top", "FIFO_IF", fifo_if);
        run_test("fifo_test");
    end

endmodule
```

Test:

```
package fifo_test_pkg;
import fifo_env_pkg::*;
import fifo_config_pkg::*;
import fifo_reset_sequence_pkg::*;
import fifo_read_sequence_pkg::*;
import fifo_write_sequence_pkg::*;
import fifo_read_write_sequence_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
    class fifo_test extends uvm_test;
        `uvm_component_utils(fifo_test)
        fifo_env env;
        fifo_reset_sequence reset_sequence;
        fifo_write_sequence write_sequence;
        fifo_read_sequence read_sequence;
        fifo_read_write_sequence read_write_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_sequence      =fifo_write_sequence::type_id::create("write_sequence");
            read_sequence       =fifo_read_sequence::type_id::create("read_sequence");
            read_write_sequence =fifo_read_write_sequence::type_id::create("read_write_sequence");
            fifo_cfg            =fifo_config::type_id::create("fifo_cfg");
            if (!uvm_config_db#(virtual FIFO_interface)::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, "*", "VIF", fifo_cfg);
        endfunction
        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            phase.raise_objection(this);
            // start the sequences
            `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_sequence starts", UVM_LOW);
            write_sequence.start(env.agt.sqr);
            `uvm_info("run_phase", "write_sequence ends", UVM_LOW);
            `uvm_info("run_phase", "read_sequence starts", UVM_LOW);
            read_sequence.start(env.agt.sqr);
            `uvm_info("run_phase", "read_sequence ends", UVM_LOW);
            `uvm_info("run_phase", "read_write_sequence starts", UVM_LOW);
            read_write_sequence.start(env.agt.sqr);
            `uvm_info("run_phase", "read_write_sequence ends", UVM_LOW);
            `uvm_info("run_phase", "Reset Asserted", UVM_LOW);
            reset_sequence.start(env.agt.sqr);
            `uvm_info("run_phase", "Reset De-asserted", UVM_LOW);
            phase.drop_objection(this);
        endtask
    endclass
endpackage
```

Reset Sequence:

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

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

        fifo_sequence_item seq_item;

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

        task body();
            seq_item = fifo_sequence_item::type_id::create("seq_item");
            start_item(seq_item);
                seq_item.rst_n   = 0;
                seq_item.rd_en   = 0;
                seq_item.wr_en   = 0;
                seq_item.data_in = 0;
            finish_item(seq_item);
        endtask
    endclass
endpackage
```

Write Sequence:

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

    class fifo_write_sequence extends uvm_sequence #(fifo_sequence_item);
        `uvm_object_utils(fifo_write_sequence)

        fifo_sequence_item seq_item;

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

        task body;
            repeat(10) begin
                seq_item=fifo_sequence_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
```

Read Sequence:

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

    class fifo_read_sequence extends uvm_sequence #(fifo_sequence_item);
        `uvm_object_utils(fifo_read_sequence)

        fifo_sequence_item seq_item;

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

        task body;
            repeat(10) begin
                seq_item=fifo_sequence_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
```

Read Write Sequence:

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

    class fifo_read_write_sequence extends uvm_sequence #(fifo_sequence_item);
        `uvm_object_utils(fifo_read_write_sequence)

        fifo_sequence_item seq_item;

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

        task body;
            repeat(5000) begin
                seq_item=fifo_sequence_item::type_id::create("seq_item");
                start_item(seq_item);
                assert(seq_item.randomize());
                finish_item(seq_item);
            end
        endtask

    endclass

endpackage
```

Environment:

```
package fifo_env_pkg;
import fifo_agent_pkg::*;
import fifo_scoreboard_pkg::*;
import fifo_coverage_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
    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);
            agt.agt_port.connect(sb.sb_export);
            agt.agt_port.connect(cov.cov_export);
        endfunction
    endclass

endpackage
```

Sequence Item:

```
package fifo_sequence_item_pkg;
import shared_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;
    logic clk;
    rand logic [FIFO_WIDTH-1:0] data_in;
    rand logic rst_n;
    rand logic wr_en;
    rand logic rd_en;
    logic [FIFO_WIDTH-1:0] data_out , data_out_ref;
    logic wr_ack, overflow, full, empty, almostfull, almostempty, underflow;


    //Inside of this class add the FIFO inputs and outputs as class variables of the class as well as adding 2 integers (RD_EN_ON_DIST & WR_EN_ON_DIST)
    int WR_EN_ON_DIST= 70;
    int RD_EN_ON_DIST= 30;

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

    ////////////////////convert2string_functions//////////

    function string convert2string();
      return $sformatf("%s rst_n = 0b%0b , data_in = 0b%0b, wr_en = 0b%0b, rd_en = 0b%0b, data_out = 0b%0b, data_out_ref = 0b%0b,
       wr_ack = 0b%0b, overflow = 0b%0b, full = 0b%0b, empty = 0b%0b, almostfull = %0b ,almostempty = 0b%0b ,underflow = 0b%0b",
        super.convert2string() ,rst_n ,data_in , wr_en, rd_en, data_out, data_out_ref, wr_ack, overflow, full, empty , almostfull, almostempty ,underflow);
    endfunction

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

    //Assert reset less often
    constraint reset_c {rst_n dist {1:/98 , 0:/2};}

    // Constraint the write enable to be high with distribution of the value WR_EN_ON_DIST & to be low with 100-WR_EN_ON_DIST
    constraint wr_en_c { wr_en dist {1 := WR_EN_ON_DIST, 0 := 100-WR_EN_ON_DIST}; }

    // Constraint the read enable the same as write enable but using RD_EN_ON_DIST
    constraint rd_en_c { rd_en dist {1 := RD_EN_ON_DIST, 0 := 100-RD_EN_ON_DIST}; }

    endclass
endpackage
```

Agent:

```
package fifo_agent_pkg;
import fifo_sequencer_pkg::*;
import fifo_config_pkg::*;
import fifo_driver_pkg::*;
import fifo_monitor_pkg::*;
import fifo_sequence_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
    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_sequence_item) agt_port;

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

        // Inside the build_phase, build the driver
        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            if(!uvm_config_db#(fifo_config)::get(this, "", "VIF", fifo_cfg))
                `uvm_fatal("build_phase","unable to get the configuration object");

            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_port = new("agt_port", this);
        endfunction

        function void connect_phase(uvm_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_port);
        endfunction

    endclass
endpackage
```

Sequencer:

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

Driver:

```
package fifo_driver_pkg;
import fifo_sequence_item_pkg::*;
import shared_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
    class fifo_driver extends uvm_driver #(fifo_sequence_item);
        `uvm_component_utils(fifo_driver)

        virtual FIFO_interface fifo_vif;

        fifo_sequence_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_sequence_item::type_id::create("stim_seq_item");
                seq_item_port.get_next_item(stim_seq_item);
                fifo_vif.rst_n   = stim_seq_item.rst_n;
                fifo_vif.rd_en   = stim_seq_item.rd_en;
                fifo_vif.wr_en   = stim_seq_item.wr_en;
                fifo_vif.data_in = stim_seq_item.data_in;
                @(negedge fifo_vif.clk);
                seq_item_port.item_done();
            end
        endtask
    endclass
endpackage
```

Monitor:

```
package fifo_monitor_pkg;
import fifo_sequence_item_pkg::*;
import shared_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
    class fifo_monitor extends uvm_monitor;
        `uvm_component_utils(fifo_monitor)

        virtual FIFO_interface fifo_vif;

        fifo_sequence_item rsp_seq_item;
        uvm_analysis_port #(fifo_sequence_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_sequence_item::type_id::create("rsp_seq_item");
                @(negedge fifo_vif.clk);

                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;

                rsp_seq_item.rst_n        = fifo_vif.rst_n;
                rsp_seq_item.wr_en        = fifo_vif.wr_en;
                rsp_seq_item.rd_en        = fifo_vif.rd_en;
                rsp_seq_item.data_in      = fifo_vif.data_in;
                mon_ap.write(rsp_seq_item);
                `uvm_info("run_phase",rsp_seq_item.convert2string(),UVM_HIGH)
            end
        endtask
    endclass
endpackage
```

Scoreboard:

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

    class fifo_scoreboard extends uvm_scoreboard;

        `uvm_component_utils(fifo_scoreboard)
        uvm_analysis_export #(fifo_sequence_item) sb_export;
        uvm_tlm_analysis_fifo #(fifo_sequence_item) sb_fifo;
        fifo_sequence_item seq_item_sb;

        logic [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 error_count=0, correct_count=0;

        logic [FIFO_WIDTH-1:0] queue[$];

        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);
                check_data(seq_item_sb);
            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= %0d, Correct Count= %0d", $time, error_count, correct_count), UVM_MEDIUM);
        endfunction


        function void check_data(fifo_sequence_item F_cd);
            reference_model(F_cd);
            outputs_check_assert;
```

```
function void check_data(fifo_sequence_item F_cd);
    reference_model(F_cd);
    outputs_check_assert:
    assert(
        data_out_ref    === F_cd.data_out    &&
        wr_ack_ref      === F_cd.wr_ack      &&
        overflow_ref    === F_cd.overflow    &&
        full_ref        === F_cd.full        &&
        empty_ref       === F_cd.empty       &&
        almostfull_ref  === F_cd.almostfull  &&
        almostempty_ref === F_cd.almostempty &&
        underflow_ref   === F_cd.underflow
    ) begin

        correct_count++;
    end
    else begin
        error_count++;
        `uvm_error("run_phase", "Comparison Error");
    end

    outputs_check_cover: cover(
        data_out_ref    === F_cd.data_out    &&
        wr_ack_ref      === F_cd.wr_ack      &&
        overflow_ref    === F_cd.overflow    &&
        full_ref        === F_cd.full        &&
        empty_ref       === F_cd.empty       &&
        almostfull_ref  === F_cd.almostfull  &&
        almostempty_ref === F_cd.almostempty &&
        underflow_ref   === F_cd.underflow
    );
endfunction


function void reference_model(fifo_sequence_item F_rm);

    if (!F_rm.rst_n) begin
        queue.delete();
        underflow_ref=0;  overflow_ref=0;
        wr_ack_ref=0;
    end
    else begin
        if ( {F_rm.wr_en,F_rm.rd_en} == 2'b01 && queue.size() != 0 ) begin
            data_out_ref=queue.pop_front();
            wr_ack_ref=0;
        end
        else if ( {F_rm.wr_en,F_rm.rd_en} == 2'b10 && queue.size() != FIFO_DEPTH ) begin
            queue.push_back(F_rm.data_in);
            wr_ack_ref=1;
        end
        else if ( {F_rm.wr_en,F_rm.rd_en} == 2'b11 ) begin
```

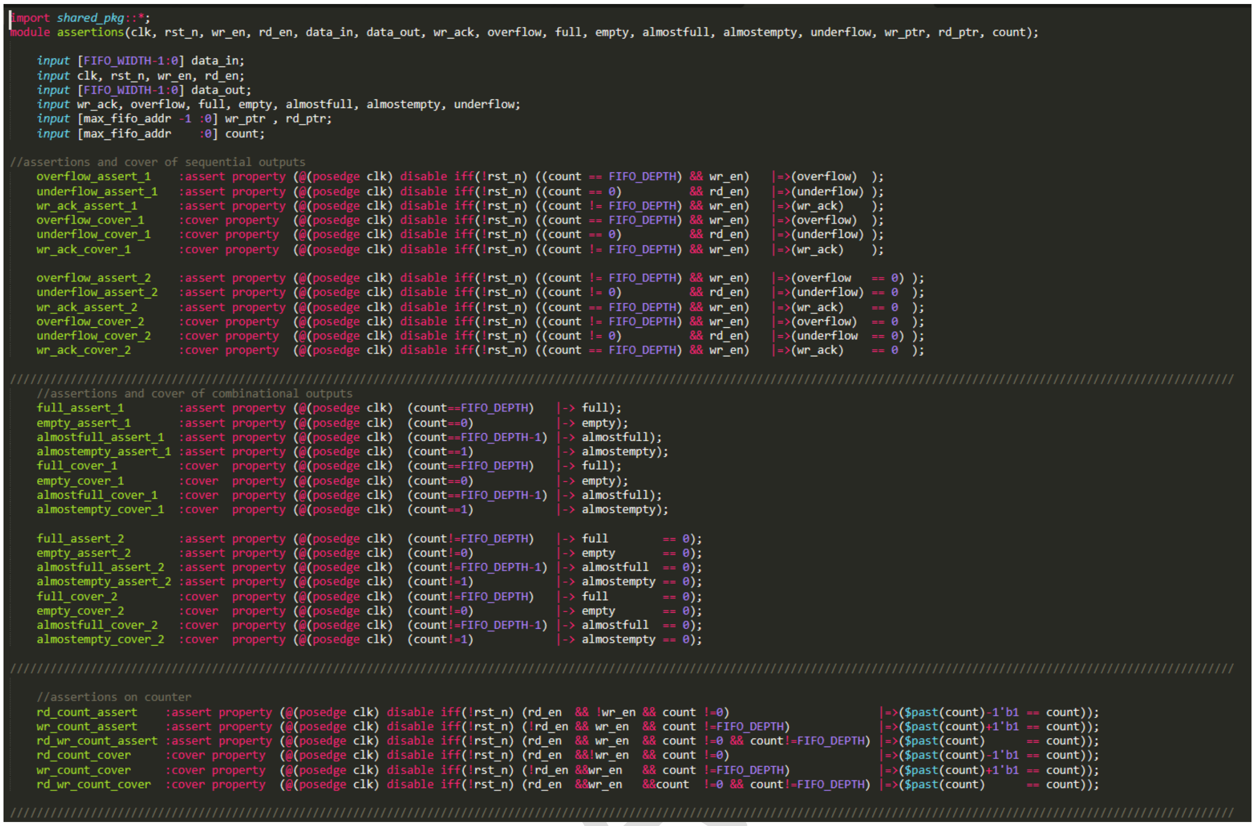
```
    function void reference_model(fifo_sequence_item F_rm);

        if (!F_rm.rst_n) begin
            queue.delete();
            underflow_ref=0;  overflow_ref=0;
            wr_ack_ref=0;
        end
        else begin
            if ( {F_rm.wr_en,F_rm.rd_en} == 2'b01 && queue.size() != 0 ) begin
                data_out_ref=queue.pop_front();
                wr_ack_ref=0;
            end
            else if ( {F_rm.wr_en,F_rm.rd_en} == 2'b10 && queue.size() != FIFO_DEPTH ) begin
                queue.push_back(F_rm.data_in);
                wr_ack_ref=1;
            end
            else if ( {F_rm.wr_en,F_rm.rd_en} == 2'b11 ) begin
                if (queue.size() == FIFO_DEPTH) begin
                    data_out_ref=queue.pop_front();
                    wr_ack_ref=0;
                end else if (queue.size() == 0) begin
                    queue.push_back(F_rm.data_in);
                    wr_ack_ref=1;
                end else begin
                    queue.push_back(F_rm.data_in);
                    wr_ack_ref=1;
                    data_out_ref=queue.pop_front();
                end
            end
            else begin
                 wr_ack_ref=0;
            end
        end

        underflow_ref   = (!F_rm.rst_n)? 0:(empty_ref && F_rm.rd_en)? 1 : 0;
        overflow_ref    = (!F_rm.rst_n)? 0:(full_ref  && F_rm.wr_en)? 1 : 0;
        full_ref        = (queue.size() == FIFO_DEPTH)? 1 : 0;
        empty_ref       = (queue.size() == 0)? 1 : 0;
        almostfull_ref  = (queue.size() == FIFO_DEPTH-1)? 1 : 0;
        almostempty_ref = (queue.size() == 1)? 1 : 0;

    endfunction

  endclass

endpackage
```

Coverage Collector:

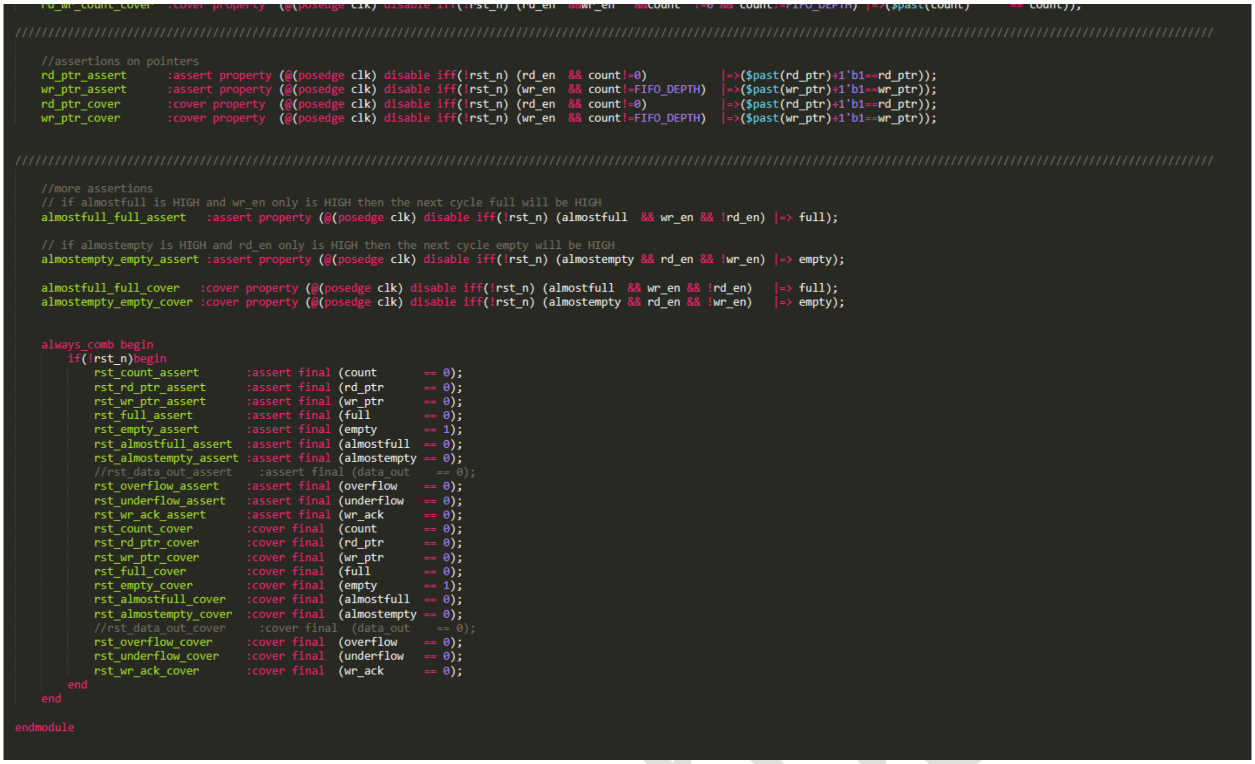
```
package fifo_coverage_pkg;
import shared_pkg::*;
import fifo_sequence_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"

class fifo_coverage extends uvm_component;
    `uvm_component_utils(fifo_coverage)
    uvm_analysis_export #(fifo_sequence_item) cov_export;
    uvm_tlm_analysis_fifo #(fifo_sequence_item) cov_fifo;
    fifo_sequence_item cov_seq_item;

    covergroup read_write_cg;
            wr_en_cp:          coverpoint cov_seq_item.wr_en       {option.weight         = 0;}
            rd_en_cp:          coverpoint cov_seq_item.rd_en       {option.weight         = 0;}
            full_cp:           coverpoint cov_seq_item.full        {bins full_HIGH        = {1}; option.weight = 0;}
            empty_cp:          coverpoint cov_seq_item.empty       {bins empty_HIGH       = {1}; option.weight = 0;}
            overflow_cp:       coverpoint cov_seq_item.overflow    {bins overflow_HIGH    = {1}; option.weight = 0;}
            underflow_cp:      coverpoint cov_seq_item.underflow   {bins underflow_HIGH   = {1}; option.weight = 0;}
            wr_ack_cp:         coverpoint cov_seq_item.wr_ack      {bins wr_ack_HIGH      = {1}; option.weight = 0;}
            almostfull_cp:     coverpoint cov_seq_item.almostfull  {bins almostfull_HIGH  = {1}; option.weight = 0;}
            almostempty_cp:    coverpoint cov_seq_item.almostempty {bins almostempty_HIGH = {1}; option.weight = 0;}


            almostfull_cross:   cross wr_en_cp, rd_en_cp, almostfull_cp;
            almostempty_cross:  cross wr_en_cp, rd_en_cp, almostempty_cp;


            // empty shouldn't be HIGH if write enable is 1 whatever read
            empty_cross:cross wr_en_cp,rd_en_cp,empty_cp iff(cov_seq_item.rst_n)          {
                illegal_bins empty_and_wr     =  binsof(wr_en_cp)intersect {1} && (binsof(rd_en_cp) intersect{0}  || binsof(rd_en_cp)  intersect{1}) && binsof(empty_cp) intersect{1};
            }
            // full shouldn't be HIGH if read enable is 1 whatever write
            full_cross:cross wr_en_cp , rd_en_cp , full_cp          {
                illegal_bins full_wr_rd       = (binsof(wr_en_cp)intersect {0} || binsof(wr_en_cp) intersect {1}) && binsof(rd_en_cp)  intersect{1} &&  binsof (full_cp) intersect{1};
            }
            // overflow shouldn't be HIGH if write enable is 0
            overflow_cross:cross wr_en_cp ,rd_en_cp, overflow_cp {
                illegal_bins wr_and_over      = binsof(wr_en_cp)intersect {0} && binsof(overflow_cp)  intersect{1};
            }
            // underflow shouldn't be HIGH if read enable is 0
            underflow_cross:cross wr_en_cp,rd_en_cp,underflow_cp {
                illegal_bins underflow_and_rd = binsof(rd_en_cp)intersect {0} && binsof(underflow_cp) intersect{1};
            }
            // write ack shouldn't be HIGH if write enable is 0
             wr_ack_cross:cross wr_en_cp , rd_en_cp , wr_ack_cp  {
                illegal_bins wr_and_wr_ack    = binsof(wr_en_cp)intersect {0} && binsof(wr_ack_cp)    intersect{1};
            }
        endgroup
```

```
            }
            // write ack shouldn't be HIGH if write enable is 0
             wr_ack_cross:cross wr_en_cp , rd_en_cp , wr_ack_cp  {
                illegal_bins wr_and_wr_ack    = binsof(wr_en_cp)intersect {0} && binsof(wr_ack
            }
        endgroup

    function new(string name = "fifo_coverage", uvm_component parent = null);
        super.new(name, parent);
        read_write_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(cov_seq_item);
            read_write_cg.sample();
        end
    endtask
endclass
endpackage
```

Assertions:



```
import shared_pkg::*;
module assertions(clk, rst_n, wr_en, rd_en, data_in, data_out, wr_ack, overflow, full, empty, almostfull, almostempty, underflow, wr_ptr, rd_ptr, count);

    input [FIFO_WIDTH-1:0] data_in;
    input clk, rst_n, wr_en, rd_en;
    input [FIFO_WIDTH-1:0] data_out;
    input wr_ack, overflow, full, empty, almostfull, almostempty, underflow;
    input [max_fifo_addr -1 :0] wr_ptr , rd_ptr;
    input [max_fifo_addr    :0] count;

//assertions and cover of sequential outputs
    overflow_assert_1    :assert property (@(posedge clk) disable iff(!rst_n) ((count == FIFO_DEPTH) && wr_en)   |=>(overflow)  );
    underflow_assert_1   :assert property (@(posedge clk) disable iff(!rst_n) ((count == 0)          && rd_en)   |=>(underflow) );
    wr_ack_assert_1      :assert property (@(posedge clk) disable iff(!rst_n) ((count != FIFO_DEPTH) && wr_en)   |=>(wr_ack)    );
    overflow_cover_1     :cover property  (@(posedge clk) disable iff(!rst_n) ((count == FIFO_DEPTH) && wr_en)   |=>(overflow)  );
    underflow_cover_1    :cover property  (@(posedge clk) disable iff(!rst_n) ((count == 0)          && rd_en)   |=>(underflow) );
    wr_ack_cover_1       :cover property  (@(posedge clk) disable iff(!rst_n) ((count != FIFO_DEPTH) && wr_en)   |=>(wr_ack)    );

    overflow_assert_2    :assert property (@(posedge clk) disable iff(!rst_n) ((count != FIFO_DEPTH) && wr_en)   |=>(overflow   == 0) );
    underflow_assert_2   :assert property (@(posedge clk) disable iff(!rst_n) ((count != 0)          && rd_en)   |=>(underflow) == 0  );
    wr_ack_assert_2      :assert property (@(posedge clk) disable iff(!rst_n) ((count == FIFO_DEPTH) && wr_en)   |=>(wr_ack)    == 0  );
    overflow_cover_2     :cover property  (@(posedge clk) disable iff(!rst_n) ((count != FIFO_DEPTH) && wr_en)   |=>(overflow)  == 0  );
    underflow_cover_2    :cover property  (@(posedge clk) disable iff(!rst_n) ((count != 0)          && rd_en)   |=>(underflow  == 0) );
    wr_ack_cover_2       :cover property  (@(posedge clk) disable iff(!rst_n) ((count == FIFO_DEPTH) && wr_en)   |=>(wr_ack)    == 0  );

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    //assertions and cover of combinational outputs
    full_assert_1        :assert property (@(posedge clk)  (count==FIFO_DEPTH)   |-> full);
    empty_assert_1       :assert property (@(posedge clk)  (count==0)            |-> empty);
    almostfull_assert_1  :assert property (@(posedge clk)  (count==FIFO_DEPTH-1) |-> almostfull);
    almostempty_assert_1 :assert property (@(posedge clk)  (count==1)            |-> almostempty);
    full_cover_1         :cover  property (@(posedge clk)  (count==FIFO_DEPTH)   |-> full);
    empty_cover_1        :cover  property (@(posedge clk)  (count==0)            |-> empty);
    almostfull_cover_1   :cover  property (@(posedge clk)  (count==FIFO_DEPTH-1) |-> almostfull);
    almostempty_cover_1  :cover  property (@(posedge clk)  (count==1)            |-> almostempty);

    full_assert_2        :assert property (@(posedge clk)  (count!=FIFO_DEPTH)   |-> full        == 0);
    empty_assert_2       :assert property (@(posedge clk)  (count!=0)            |-> empty       == 0);
    almostfull_assert_2  :assert property (@(posedge clk)  (count!=FIFO_DEPTH-1) |-> almostfull  == 0);
    almostempty_assert_2 :assert property (@(posedge clk)  (count!=1)            |-> almostempty == 0);
    full_cover_2         :cover  property (@(posedge clk)  (count!=FIFO_DEPTH)   |-> full        == 0);
    empty_cover_2        :cover  property (@(posedge clk)  (count!=0)            |-> empty       == 0);
    almostfull_cover_2   :cover  property (@(posedge clk)  (count!=FIFO_DEPTH-1) |-> almostfull  == 0);
    almostempty_cover_2  :cover  property (@(posedge clk)  (count!=1)            |-> almostempty == 0);

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

    //assertions on counter
    rd_count_assert    :assert property (@(posedge clk) disable iff(!rst_n) (rd_en  && !wr_en && count !=0)                      |=>($past(count)-1'b1 == count));
    wr_count_assert    :assert property (@(posedge clk) disable iff(!rst_n) (!rd_en && wr_en  && count !=FIFO_DEPTH)             |=>($past(count)+1'b1 == count));
    rd_wr_count_assert :assert property (@(posedge clk) disable iff(!rst_n) (rd_en  && wr_en  && count !=0 && count!=FIFO_DEPTH) |=>($past(count)      == count));
    rd_count_cover     :cover property  (@(posedge clk) disable iff(!rst_n) (rd_en  &&!wr_en  && count !=0)                      |=>($past(count)-1'b1 == count));
    wr_count_cover     :cover property  (@(posedge clk) disable iff(!rst_n) (!rd_en &&wr_en   && count !=FIFO_DEPTH)             |=>($past(count)+1'b1 == count));
    rd_wr_count_cover  :cover property  (@(posedge clk) disable iff(!rst_n) (rd_en  &&wr_en   &&count  !=0 && count!=FIFO_DEPTH) |=>($past(count)      == count));

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```



```
    rd_wr_count_cover  :cover property  (@(posedge clk) disable iff(!rst_n) (rd_en  &&wr_en   &&count  !=0 && count!=FIFO_DEPTH) |=>($past(count)      == count));
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

    //assertions on pointers
    rd_ptr_assert      :assert property (@(posedge clk) disable iff(!rst_n) (rd_en  && count!=0)           |=>($past(rd_ptr)+1'b1==rd_ptr));
    wr_ptr_assert      :assert property (@(posedge clk) disable iff(!rst_n) (wr_en  && count!=FIFO_DEPTH)  |=>($past(wr_ptr)+1'b1==wr_ptr));
    rd_ptr_cover       :cover property  (@(posedge clk) disable iff(!rst_n) (rd_en  && count!=0)           |=>($past(rd_ptr)+1'b1==rd_ptr));
    wr_ptr_cover       :cover property  (@(posedge clk) disable iff(!rst_n) (wr_en  && count!=FIFO_DEPTH)  |=>($past(wr_ptr)+1'b1==wr_ptr));


//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

    //more assertions
    // if almostfull is HIGH and wr_en only is HIGH then the next cycle full will be HIGH
    almostfull_full_assert   :assert property (@(posedge clk) disable iff(!rst_n) (almostfull  && wr_en && !rd_en) |=> full);

    // if almostempty is HIGH and rd_en only is HIGH then the next cycle empty will be HIGH
    almostempty_empty_assert :assert property (@(posedge clk) disable iff(!rst_n) (almostempty && rd_en && !wr_en) |=> empty);

    almostfull_full_cover   :cover property (@(posedge clk) disable iff(!rst_n) (almostfull  && wr_en && !rd_en)   |=> full);
    almostempty_empty_cover :cover property (@(posedge clk) disable iff(!rst_n) (almostempty && rd_en && !wr_en)   |=> empty);


    always_comb begin
        if(!rst_n)begin
            rst_count_assert       :assert final (count       == 0);
            rst_rd_ptr_assert      :assert final (rd_ptr      == 0);
            rst_wr_ptr_assert      :assert final (wr_ptr      == 0);
            rst_full_assert        :assert final (full        == 0);
            rst_empty_assert       :assert final (empty       == 1);
            rst_almostfull_assert  :assert final (almostfull  == 0);
            rst_almostempty_assert :assert final (almostempty == 0);
            //rst_data_out_assert    :assert final (data_out    == 0);
            rst_overflow_assert    :assert final (overflow    == 0);
            rst_underflow_assert   :assert final (underflow   == 0);
            rst_wr_ack_assert      :assert final (wr_ack      == 0);
            rst_count_cover        :cover final  (count       == 0);
            rst_rd_ptr_cover       :cover final  (rd_ptr      == 0);
            rst_wr_ptr_cover       :cover final  (wr_ptr      == 0);
            rst_full_cover         :cover final  (full        == 0);
            rst_empty_cover        :cover final  (empty       == 1);
            rst_almostfull_cover   :cover final  (almostfull  == 0);
            rst_almostempty_cover  :cover final  (almostempty == 0);
            //rst_data_out_cover     :cover final  (data_out    == 0);
            rst_overflow_cover     :cover final  (overflow    == 0);
            rst_underflow_cover    :cover final  (underflow   == 0);
            rst_wr_ack_cover       :cover final  (wr_ack      == 0);
        end
    end

endmodule
```

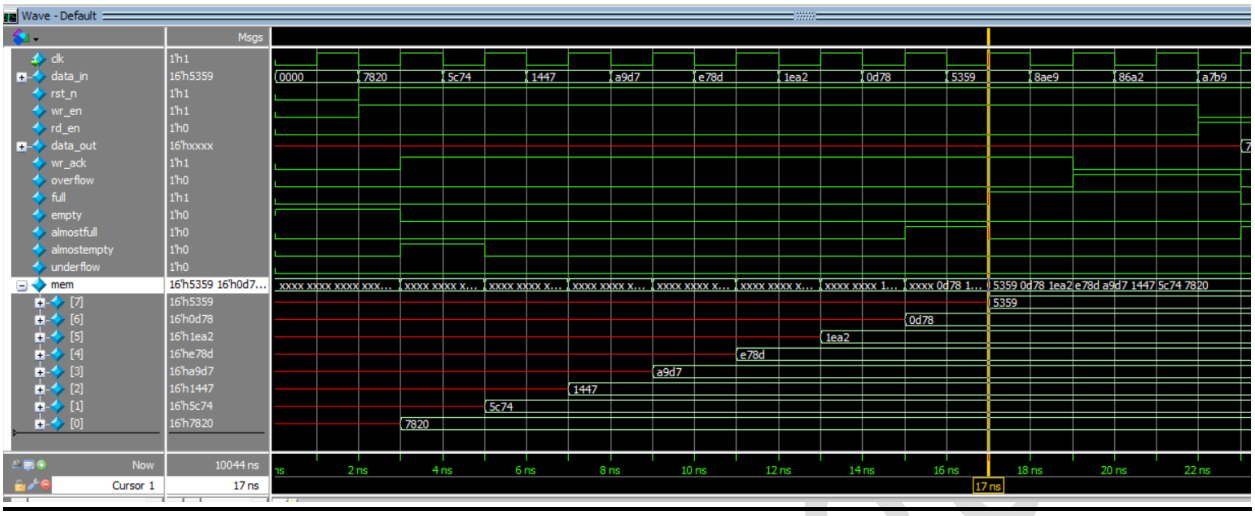
Source Files:

```
shared_pkg.sv
interface.sv
FIFO.sv
assertions.sv
sequence_item.sv
sequencer.sv
reset_sequence.sv
read_sequence.sv
write_sequence.sv
read_write_sequence.sv
config.sv
driver.sv
monitor.sv
agent.sv
coverage.sv
scoreboard.sv
env.sv
test.sv
top.sv
```

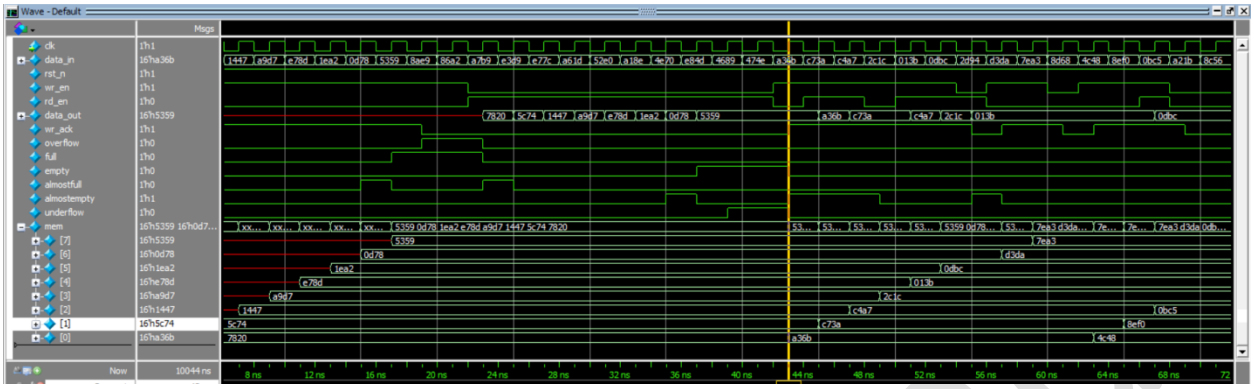
Do File:

```
vlib work
vlog -f src_files.list +cover -covercells
vsim -voptargs=+acc work.top -cover
add wave /top/fifo_if/*
coverage save top.ucdb -onexit
run -all
```

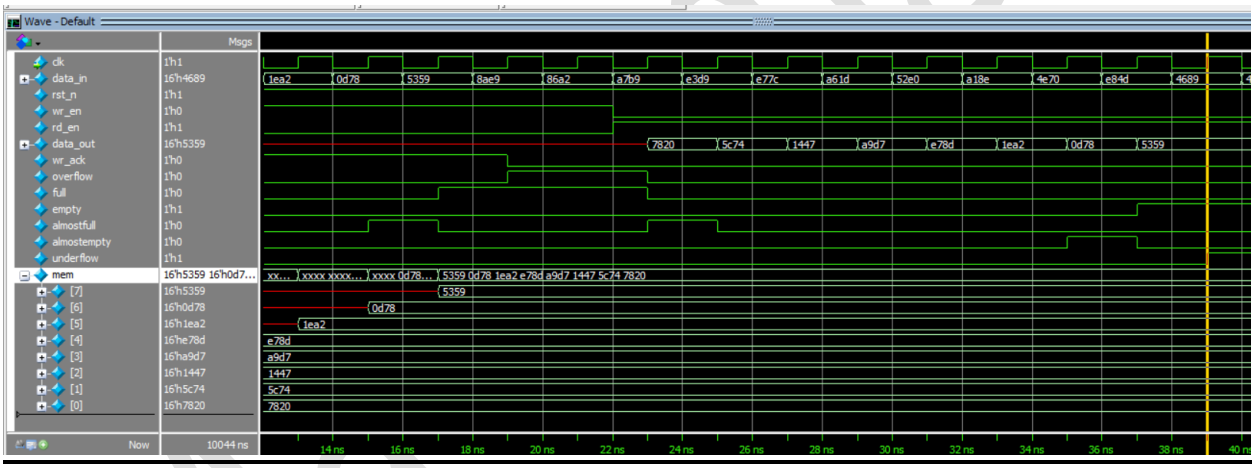
Write Only:



Read Only:



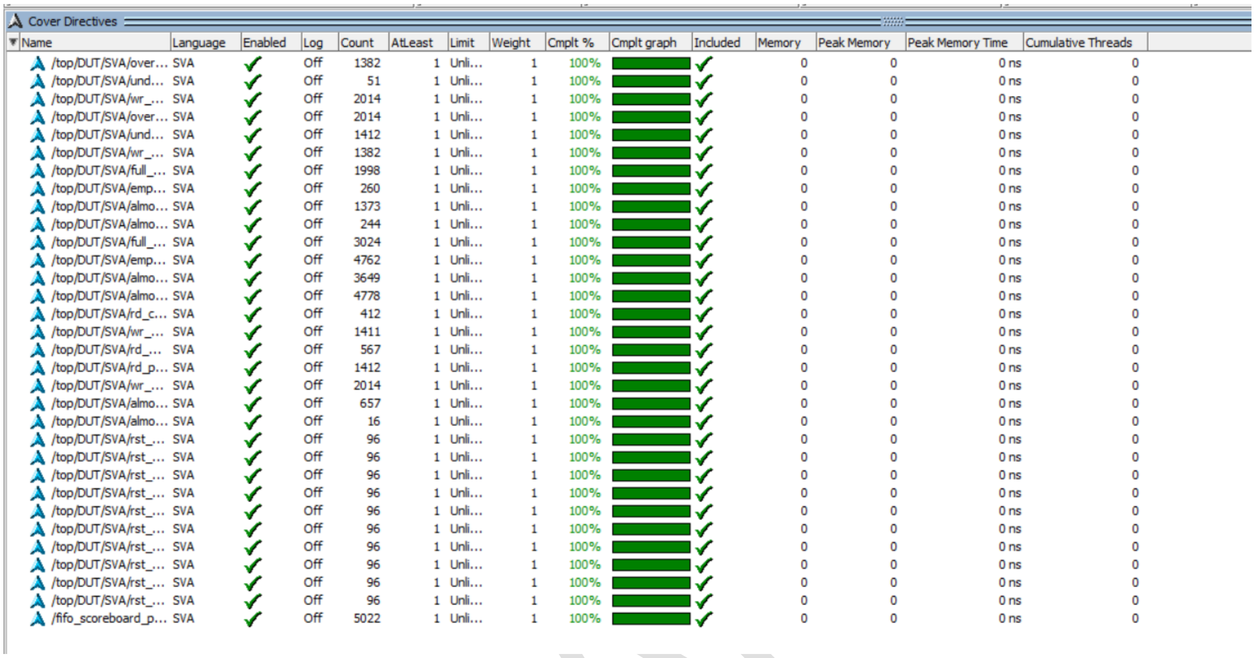
After that: all is read write



UVM report summary:

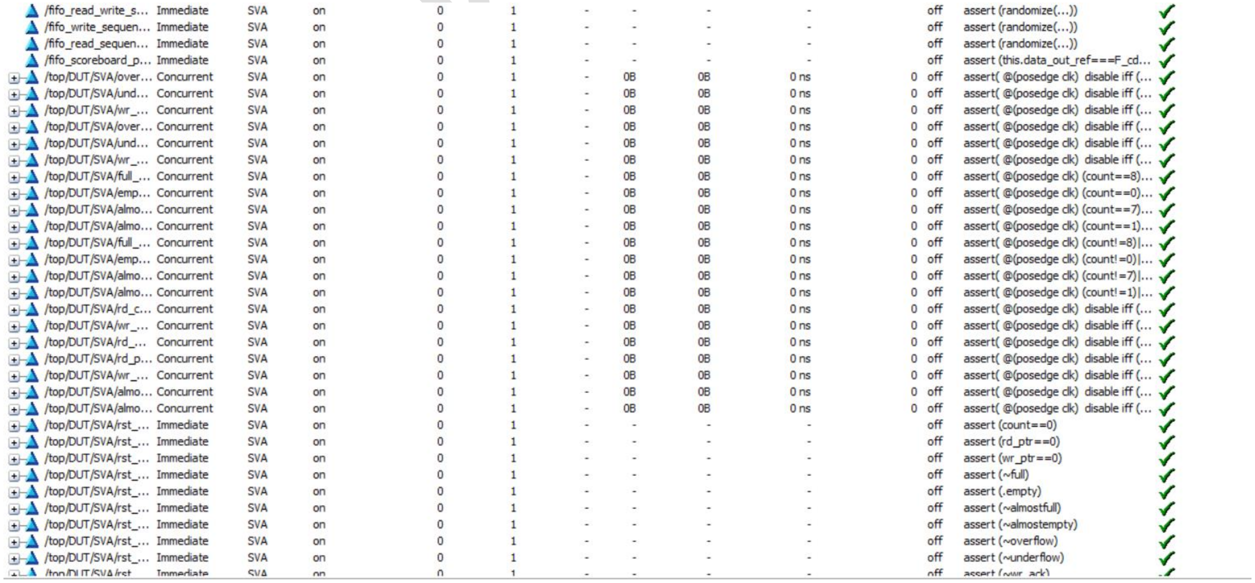
```
# UVM_INFO D:/Digital_Verfication_Using_SV&UVM/Mark_Amgad_UVM_FIFO_Project/test.sv(43) @ 0: uvm_test_top [run_phase] Reset Asserted
# UVM_INFO D:/Digital_Verfication_Using_SV&UVM/Mark_Amgad_UVM_FIFO_Project/test.sv(45) @ 2: uvm_test_top [run_phase] Reset De-asserted
# UVM_INFO D:/Digital_Verfication_Using_SV&UVM/Mark_Amgad_UVM_FIFO_Project/test.sv(46) @ 2: uvm_test_top [run_phase] write_sequence starts
# UVM_INFO D:/Digital_Verfication_Using_SV&UVM/Mark_Amgad_UVM_FIFO_Project/test.sv(48) @ 22: uvm_test_top [run_phase] write_sequence ends
# UVM_INFO D:/Digital_Verfication_Using_SV&UVM/Mark_Amgad_UVM_FIFO_Project/test.sv(49) @ 22: uvm_test_top [run_phase] read_sequence starts
# UVM_INFO D:/Digital_Verfication_Using_SV&UVM/Mark_Amgad_UVM_FIFO_Project/test.sv(51) @ 42: uvm_test_top [run_phase] read_sequence ends
# UVM_INFO D:/Digital_Verfication_Using_SV&UVM/Mark_Amgad_UVM_FIFO_Project/test.sv(52) @ 42: uvm_test_top [run_phase] read_write_sequence starts
# UVM_INFO D:/Digital_Verfication_Using_SV&UVM/Mark_Amgad_UVM_FIFO_Project/test.sv(54) @ 10042: uvm_test_top [run_phase] read_write_sequence ends
# UVM_INFO D:/Digital_Verfication_Using_SV&UVM/Mark_Amgad_UVM_FIFO_Project/test.sv(55) @ 10042: uvm_test_top [run_phase] Reset Asserted
# UVM_INFO D:/Digital_Verfication_Using_SV&UVM/Mark_Amgad_UVM_FIFO_Project/test.sv(57) @ 10044: uvm_test_top [run_phase] Reset De-asserted
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 10044: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO D:/Digital_Verfication_Using_SV&UVM/Mark_Amgad_UVM_FIFO_Project/scoreboard.sv(46) @ 10044: uvm_test_top.env.sb [report_phase] At time 10044: Simulation Ends and Error Count= 0, Correct Count= 5022
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   15
# 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]    10
# ** Note: $finish    : C:/MentorGraphics/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 10044 ns  Iteration: 61  Instance: /top
# 1
# Break in Task uvm_pkg/uvm_root::run_test at C:/MentorGraphics/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh line 430
```

Cover Directives & Assertions:



Cover Directives

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
/top/DUT/SVA/over...	SVA	✓	Off	1382	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/top/DUT/SVA/und...	SVA	✓	Off	51	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/top/DUT/SVA/wr_...	SVA	✓	Off	2014	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/top/DUT/SVA/over...	SVA	✓	Off	2014	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/top/DUT/SVA/und...	SVA	✓	Off	1412	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/top/DUT/SVA/wr_...	SVA	✓	Off	1382	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/top/DUT/SVA/full_...	SVA	✓	Off	1998	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/top/DUT/SVA/emp...	SVA	✓	Off	260	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/top/DUT/SVA/almo...	SVA	✓	Off	1373	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/top/DUT/SVA/almo...	SVA	✓	Off	244	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/top/DUT/SVA/full_...	SVA	✓	Off	3024	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/top/DUT/SVA/emp...	SVA	✓	Off	4762	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/top/DUT/SVA/almo...	SVA	✓	Off	3649	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/top/DUT/SVA/almo...	SVA	✓	Off	4778	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/top/DUT/SVA/rd_c...	SVA	✓	Off	412	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/top/DUT/SVA/wr_...	SVA	✓	Off	1411	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/top/DUT/SVA/rd_...	SVA	✓	Off	567	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/top/DUT/SVA/rd_p...	SVA	✓	Off	1412	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/top/DUT/SVA/wr_...	SVA	✓	Off	2014	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/top/DUT/SVA/almo...	SVA	✓	Off	657	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/top/DUT/SVA/almo...	SVA	✓	Off	16	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/top/DUT/SVA/rst_...	SVA	✓	Off	96	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/top/DUT/SVA/rst_...	SVA	✓	Off	96	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/top/DUT/SVA/rst_...	SVA	✓	Off	96	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/top/DUT/SVA/rst_...	SVA	✓	Off	96	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/top/DUT/SVA/rst_...	SVA	✓	Off	96	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/top/DUT/SVA/rst_...	SVA	✓	Off	96	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/top/DUT/SVA/rst_...	SVA	✓	Off	96	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/top/DUT/SVA/rst_...	SVA	✓	Off	96	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/top/DUT/SVA/rst_...	SVA	✓	Off	96	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/top/DUT/SVA/rst_...	SVA	✓	Off	96	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/fifo_scoreboard_p...	SVA	✓	Off	5022	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0



/fifo_read_write_s...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/fifo_write_sequen...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/fifo_read_sequen...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/fifo_scoreboard_p...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (this.data_out_ref===F_cd...	✓
/top/DUT/SVA/over...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (...	✓
/top/DUT/SVA/und...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (...	✓
/top/DUT/SVA/wr_...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (...	✓
/top/DUT/SVA/over...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (...	✓
/top/DUT/SVA/und...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (...	✓
/top/DUT/SVA/wr_...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (...	✓
/top/DUT/SVA/full_...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) (count==8)...	✓
/top/DUT/SVA/emp...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) (count==0)...	✓
/top/DUT/SVA/almo...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) (count==7)...	✓
/top/DUT/SVA/almo...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) (count==1)...	✓
/top/DUT/SVA/full_...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) (count!=8)|...	✓
/top/DUT/SVA/emp...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) (count!=0)|...	✓
/top/DUT/SVA/almo...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) (count!=7)|...	✓
/top/DUT/SVA/almo...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) (count!=1)|...	✓
/top/DUT/SVA/rd_c...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (...	✓
/top/DUT/SVA/wr_...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (...	✓
/top/DUT/SVA/rd_...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (...	✓
/top/DUT/SVA/rd_p...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (...	✓
/top/DUT/SVA/wr_...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (...	✓
/top/DUT/SVA/almo...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (...	✓
/top/DUT/SVA/almo...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge clk) disable iff (...	✓
/top/DUT/SVA/rst_...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (count==0)	✓
/top/DUT/SVA/rst_...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (rd_ptr==0)	✓
/top/DUT/SVA/rst_...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (wr_ptr==0)	✓
/top/DUT/SVA/rst_...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (~full)	✓
/top/DUT/SVA/rst_...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (.empty)	✓
/top/DUT/SVA/rst_...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (~almostfull)	✓
/top/DUT/SVA/rst_...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (~almostempty)	✓
/top/DUT/SVA/rst_...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (~overflow)	✓
/top/DUT/SVA/rst_...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (~underflow)	✓
/top/DUT/SVA/rst	Immediate	SVA	on	0	1	-	-	-	-		off	assert (~wr_ack)	✓

Covergroup :

Covergroups

Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment
/fifo_coverage_pk...		100.00%							
TYPE read_wri...		100.00%	100	100.00...		✓	auto(1)		
CVP read_...		100.00%	100	100.00...		✓			
CVP read_...		100.00%	100	100.00...		✓			
CVP read_...		100.00%	100	100.00...		✓			
CVP read_...		100.00%	100	100.00...		✓			
CVP read_...		100.00%	100	100.00...		✓			
CVP read_...		100.00%	100	100.00...		✓			
CVP read_...		100.00%	100	100.00...		✓			
CVP read_...		100.00%	100	100.00...		✓			
CROSS rea...		100.00%	100	100.00...		✓			
CROSS rea...		100.00%	100	100.00...		✓			
CROSS rea...		100.00%	100	100.00...		✓			
CROSS rea...		100.00%	100	100.00...		✓			
CROSS rea...		100.00%	100	100.00...		✓			
CROSS rea...		100.00%	100	100.00...		✓			
CROSS rea...		100.00%	100	100.00...		✓			

Function Coverage report:

```
================================================================================
=== Instance: /top/DUT/SVA
=== Design Unit: work.assertions
================================================================================

Directive Coverage:
    Directives                      31        31         0   100.00%
```

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

Covergroup Coverage:
    Covergroups                      1        na        na   100.00%
        Coverpoints/Crosses         16        na        na        na
            Covergroup Bins         29        29         0   100.00%
```

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

Directive Coverage:
    Directives                       1         1         0   100.00%
```

```
TOTAL COVERGROUP COVERAGE: 100.00%  COVERGROUP TYPES: 1
```

```
TOTAL DIRECTIVE COVERAGE: 100.00%  COVERS: 32

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

Fuction_cover_report.txt

Assertions report:

```
================================================================================
=== Instance: /top/DUT/SVA
=== Design Unit: work.assertions
================================================================================

Assertion Coverage:
    Assertions                      31        31         0   100.00%
----------------------------------------------------------------------
                                                     0         1

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

Code Coverage report:

```
================================================================================
=== Instance: /top/DUT
=== Design Unit: work.FIFO
================================================================================
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        27        27         0   100.00%

================================Branch Details================================

Branch Coverage for instance /top/DUT

    Line         Item                      Count     Source
    ----         ----                      -----     ------
  File FIFO.sv
------------------------------------IF Branch------------------------------------
    15                                      5116     Count coming in to IF
    15             1                         191                if (!rst_n) begin
    20             1                        2054                else if (wr_en && count < FIFO_DEPTH) begin
    26             1                        2871                else begin
Branch totals: 3 hits of 3 branches = 100.00%

------------------------------------IF Branch------------------------------------
    28                                      2871     Count coming in to IF
    28             1                        1415                        if (wr_en && full)
    30             1                        1456                        else
Branch totals: 2 hits of 2 branches = 100.00%

------------------------------------IF Branch------------------------------------
    36                                      3757     Count coming in to IF
    36             1                         190                if (!rst_n) begin
    41             1                        1440                else if (rd_en && count != 0) begin
    46             1                        2127                else begin
Branch totals: 3 hits of 3 branches = 100.00%

------------------------------------IF Branch------------------------------------
    48                                      2127     Count coming in to IF
    48             1                          50                        if (rd_en && empty)
    50             1                        2077                        else
Branch totals: 2 hits of 2 branches = 100.00%

------------------------------------IF Branch------------------------------------
    57                                      4320     Count coming in to IF
    57             1                         191                if (!rst_n) begin
    60             1                        4129                else begin
Branch totals: 2 hits of 2 branches = 100.00%

------------------------------------IF Branch------------------------------------
    61                                      4129     Count coming in to IF
    61             1                        1436                        if          ( ({wr_en, rd_en} == 2'b10) && !full)
    64             1                         420                        else if ( ({wr_en, rd_en} == 2'b01) && !empty)
    67             1                         932                        else if   ({wr_en, rd_en} == 2'b11) begin
                                            1341     All False Count
Branch totals: 4 hits of 4 branches = 100.00%

------------------------------------IF Branch------------------------------------
    69                                       932     Count coming in to IF
    69             1                         438                                if (full)
    71             1                          36                                else if (empty)
```

```
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                        106       106         0   100.00%

================================Toggle Details================================

Toggle Coverage for instance /top/DUT --

                                            Node      1H->0L      0L->1H  "Coverage"
                                            ----------------------------------------
                                     almostempty           1           1      100.00
                                      almostfull           1           1      100.00
                                             clk           1           1      100.00
                                        count[3-0]         1           1      100.00
                                     data_in[0-15]         1           1      100.00
                                    data_out[15-0]         1           1      100.00
                                           empty           1           1      100.00
                                            full           1           1      100.00
                                        overflow           1           1      100.00
                                           rd_en           1           1      100.00
                                     rd_ptr[2-0]           1           1      100.00
                                           rst_n           1           1      100.00
                                       underflow           1           1      100.00
                                          wr_ack           1           1      100.00
                                           wr_en           1           1      100.00
                                     wr_ptr[2-0]           1           1      100.00

Total Node Count     =         53
Toggled Node Count   =         53
Untoggled Node Count =          0

Toggle Coverage      =     100.00% (106 of 106 bins)

================================================================================
=== Instance: /top
=== Design Unit: work.top
================================================================================
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                       6         6         0   100.00%
```

```
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                      29        29         0   100.00%
```

```
    81                                    2425     Count coming in to IF
    81             1                       111        assign empty       = (count == 0)           ? 1 : 0;
    81             2                      2314        assign empty       = (count == 0)           ? 1 : 0;
Branch totals: 2 hits of 2 branches = 100.00%

-----------------------------------IF Branch-----------------------------------
    82                                    2425     Count coming in to IF
    82             1                       808        assign almostfull  = (count == FIFO_DEPTH-1) ? 1 : 0;
    82             2                      1617        assign almostfull  = (count == FIFO_DEPTH-1) ? 1 : 0;
Branch totals: 2 hits of 2 branches = 100.00%

-----------------------------------IF Branch-----------------------------------
    83                                    2425     Count coming in to IF
    83             1                       136        assign almostempty = (count == 1)           ? 1 : 0;
    83             2                      2289        assign almostempty = (count == 1)           ? 1 : 0;
Branch totals: 2 hits of 2 branches = 100.00%


Condition Coverage:
    Enabled Coverage              Bins    Covered    Misses  Coverage
    ----------------              ----    ----       ------  --------
    Conditions                      20        18         2    90.00%
```