Digital Verification

Final Project

coverage_fifo_uvm.text includes all reports and its included in the compressed fil

Bugs report:

1. Underflow is sequential output. So, it should be defined as reg and be assigned inside always block.

2. Handling resetting of signals for each output logic.

3. Overflow is deasserted once a successful write is done. So, as underflow in case of reading.

4. Case no write occurs wr_ack is deasserted.

5. Added logic of the case where both read and write enables are asserted.

6. Almostfull flag is asserted case count = FIXED_WIDTH -1 not -2

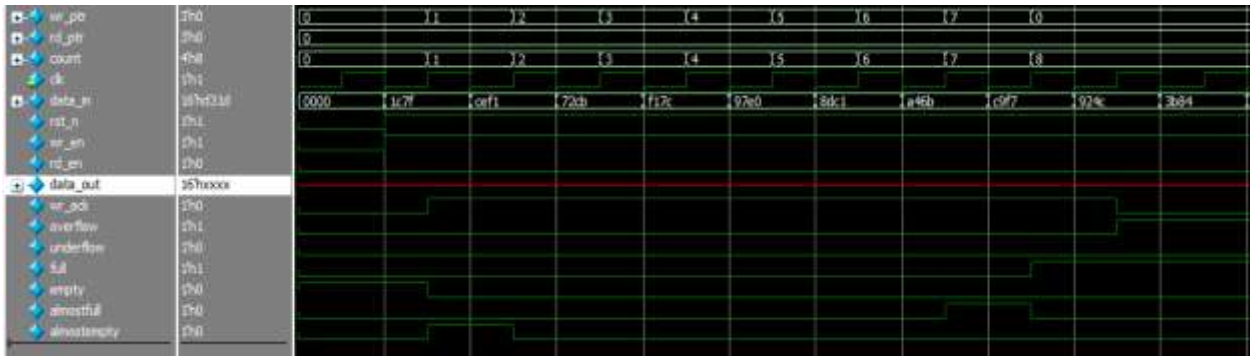
Assertions table:

Feature	Assertion
checks that after a reset (`rst_n` is low), the write pointer (`wr_ptr`), read pointer (`rd_ptr`), and count are all reset to zero	**if (!inter.rst_n) begin reset:assert final((!inter.wr_ptr)&&(!inter.rd_ptr)&&(!inter.count)); end**
checks that the `full` signal is asserted when the FIFO count reaches its depth (FIFO is full).	**fulla:assert final(inter.full == (inter.count == inter.FIFO_DEPTH)? 1:0);**
checks that the `almostfull` signal is asserted when the FIFO is one entry short of being full.	**almostfulla:assert final(inter.almostfull == (inter.count == inter.FIFO_DEPTH - 1)? 1:0);**
checks that the `empty` signal is asserted when the FIFO count is zero (FIFO is empty).	**emptya:assert final(inter.empty == (inter.count == 0)? 1:0);**
checks that the `almostempty` signal is asserted when the FIFO count is one (FIFO is almost empty).	**almostemptya:assert final(inter.almostempty == (inter.count == 1)? 1:0);**
ensures that when the `wr_en` signal is	**property check_wr_ack;**

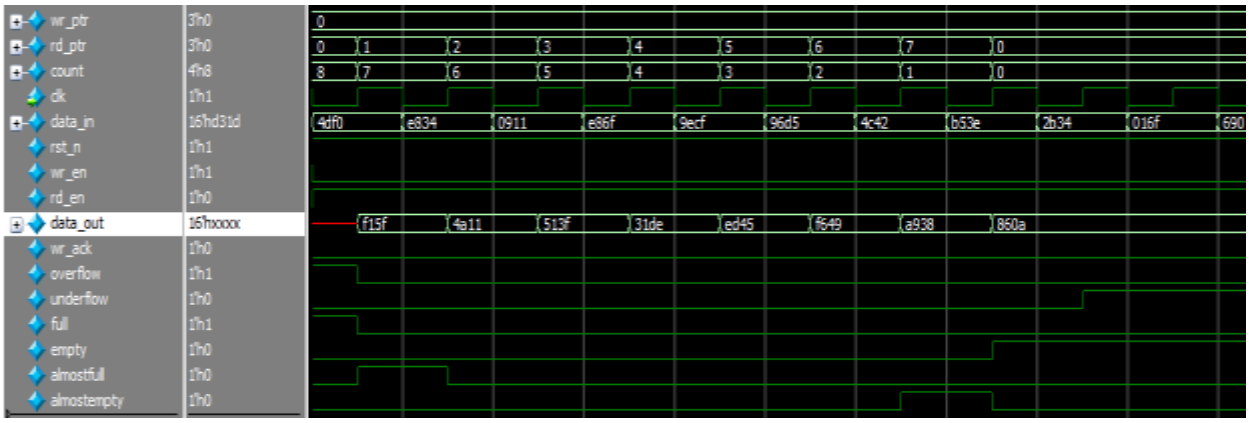
asserted (write is enabled), and the FIFO is not full, the write acknowledgment (`wr_ack`) is asserted on the next clock cycle.	**@(posedge inter.clk) disable iff (!inter.rst_n) ($past(inter.wr_en) && ($past(inter.count) < inter.FIFO_DEPTH) && $past(inter.rst_n)|-> inter.wr_ack == 1); endproperty**
checks that an overflow occurs if a write is attempted when the FIFO is full.	**property check_overflow; @(posedge inter.clk) disable iff (!inter.rst_n) ($past(inter.wr_en) && $past(inter.full)&& $past(inter.rst_n)) |-> inter.overflow; endproperty**
checks that an underflow occurs if a read is attempted when the FIFO is empty.	**property check_underflow; @(posedge inter.clk) disable iff (!inter.rst_n) ($past(inter.rd_en) && $past(inter.empty) && $past(inter.rst_n) |-> inter.underflow); endproperty**
checks that the FIFO count remains unchanged if both read and write are enabled at the same time	**property check_count_stable; @(posedge inter.clk) disable iff (!inter.rst_n) ($past(inter.wr_en) && $past(inter.rd_en) && !$past(inter.full) && !$past(inter.empty) && $past(inter.rst_n)) |-> (inter.count == $past(inter.count)); endproperty**
checks that the FIFO count is incremented by 1 when a write occurs and no read is happening.	**property check_count_increment; @(posedge inter.clk) disable iff (!inter.rst_n) ($past(inter.wr_en) && !$past(inter.rd_en) && !$past(inter.full) && $past(inter.rst_n)) |-> (inter.count == $past(inter.count)+ 1'b1); endproperty**
checks that the FIFO count is decremented by 1 when a read occurs and no write is happening.	**property check_count_decrement; @(posedge inter.clk) disable iff (!inter.rst_n) (!$past(inter.wr_en) && $past(inter.rd_en) && !$past(inter.empty) && $past(inter.rst_n)) |-> (inter.count == $past(inter.count) - 1'b1); endproperty**
checks that the write pointer (`wr_ptr`) is incremented by 1 when a write occurs, and the FIFO is not full.	**property check_wr_ptr_increment; @(posedge inter.clk) disable iff (!inter.rst_n) ($past(inter.wr_en) && ($past(inter.count) < inter.FIFO_DEPTH) && $past(inter.rst_n)|-> (inter.wr_ptr == $past(inter.wr_ptr) + 1'b1)); endproperty**
checks that the read pointer (`rd_ptr`) is incremented by 1 when a read occurs, and the FIFO is not empty.	**property check_rd_ptr_increment; @(posedge inter.clk) disable iff (!inter.rst_n) ($past(inter.rd_en) && ($past(inter.count) != 0) && $past(inter.rst_n)|-> (inter.rd_ptr == $past(inter.rd_ptr) + 1'b1));**

Waveform snippets:

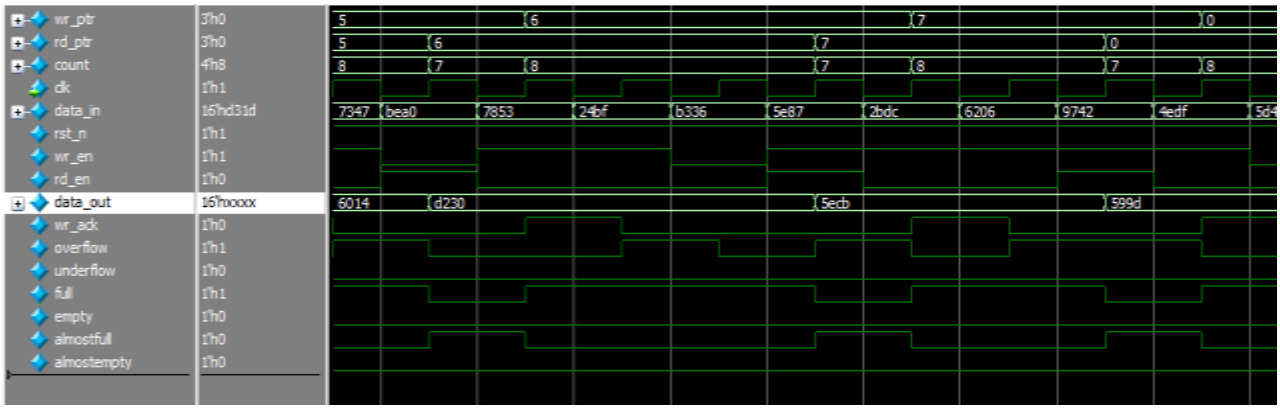
Reset &Write sequences:



Read sequence:



Write_read_sequence:



Code coverage snippets:

```
================================================================================
=== File: FIFO.sv
================================================================================
Statement Coverage:
    Enabled Coverage            Active      Hits    Misses % Covered
    ----------------            ------      ----    ------ ---------
    Stmts                           30        30         0     100.0

=================================Statement Details================================

Statement Coverage for file FIFO.sv --

    1                                           //vcover report top_FIFO.ucdb -all -details -output coverage_fifo.txt
    2                                           module FIFO(FIFO_interface.DUT inter);
    3
    4                                             // Write Logic
    5               1                     10419   always @(posedge inter.clk or negedge inter.rst_n) begin
    6                                               if (!inter.rst_n) begin
    7               1                       441       inter.wr_ptr <= 0;
    8                                                 // Bug reset of signals
    9               1                       441       inter.wr_ack <= 0;
    10              1                       441       inter.overflow <= 0;
    11                                              end else if (inter.wr_en && inter.count < inter.FIFO_DEPTH) begin
    12                                                // Write operation
    13              1                      4085       inter.mem[inter.wr_ptr] <= inter.data_in;
    14              1                      4085       inter.wr_ack <= 1;
    15              1                      4085       inter.wr_ptr <= inter.wr_ptr + 1;
    16                                                // Bug successful write inter.overflow not handled
    17              1                      4085       inter.overflow <= 0;
    18                                              end else begin
    19                                                // Bug Reset inter.wr_ack when no write occurs
    20              1                      5893       inter.wr_ack <= 0;
    21                                                if (inter.full && inter.wr_en) begin
    22              1                      2875         inter.overflow <= 1;
<
                                                                                                        Ln 1, Co
```

```
Branch Coverage:
    Enabled Coverage            Active      Hits    Misses % Covered
    ----------------            ------      ----    ------ ---------
    Branches                        27        27         0     100.0

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

Branch Coverage for file FIFO.sv --

------------------------------------IF Branch------------------------------------
    6                                       10419     Count coming in to IF
    6               1                         441         if (!inter.rst_n) begin
    11              1                        4085         end else if (inter.wr_en && inter.count < inter.FIFO_DEPTH) begin
    18              1                        5893         end else begin
Branch totals: 3 hits of 3 branches = 100.0%

------------------------------------IF Branch------------------------------------
    21                                       5893     Count coming in to IF
    21              1                        2875           if (inter.full && inter.wr_en) begin
    23              1                        3018           end else begin
Branch totals: 2 hits of 2 branches = 100.0%

------------------------------------IF Branch------------------------------------
    31                                      10419     Count coming in to IF
    31              1                         441         if (!inter.rst_n) begin
    35              1                        2747         end else if (inter.rd_en && inter.count != 0) begin
    40              1                        7231         end else begin
Branch totals: 3 hits of 3 branches = 100.0%

------------------------------------IF Branch------------------------------------
    41                                       7231     Count coming in to IF
    41              1                         202           if (inter.empty && inter.rd_en) begin
    43              1                        7029           end else begin
Branch totals: 2 hits of 2 branches = 100.0%
<
```

Functional coverage:

/fifo_coverage_pk…		100.0%			
TYPE cvr_grou…	fifo_coverage	100.0%	100	100.0%	✓
CVP cvr_gr…	fifo_coverage	100.0%	100	100.0%	✓
CVP cvr_gr…	fifo_coverage	100.0%	100	100.0%	✓
CVP cvr_gr…	fifo_coverage	100.0%	100	100.0%	✓
CVP cvr_gr…	fifo_coverage	100.0%	100	100.0%	✓
CVP cvr_gr…	fifo_coverage	100.0%	100	100.0%	✓
CVP cvr_gr…	fifo_coverage	100.0%	100	100.0%	✓
CVP cvr_gr…	fifo_coverage	100.0%	100	100.0%	✓
CVP cvr_gr…	fifo_coverage	100.0%	100	100.0%	✓
CVP cvr_gr…	fifo_coverage	100.0%	100	100.0%	✓
CVP cvr_gr…	fifo_coverage	100.0%	100	100.0%	✓
CROSS cvr…	fifo_coverage	100.0%	100	100.0%	✓
CROSS cvr…	fifo_coverage	100.0%	100	100.0%	✓
CROSS cvr…	fifo_coverage	100.0%	100	100.0%	✓
CROSS cvr…	fifo_coverage	100.0%	100	100.0%	✓

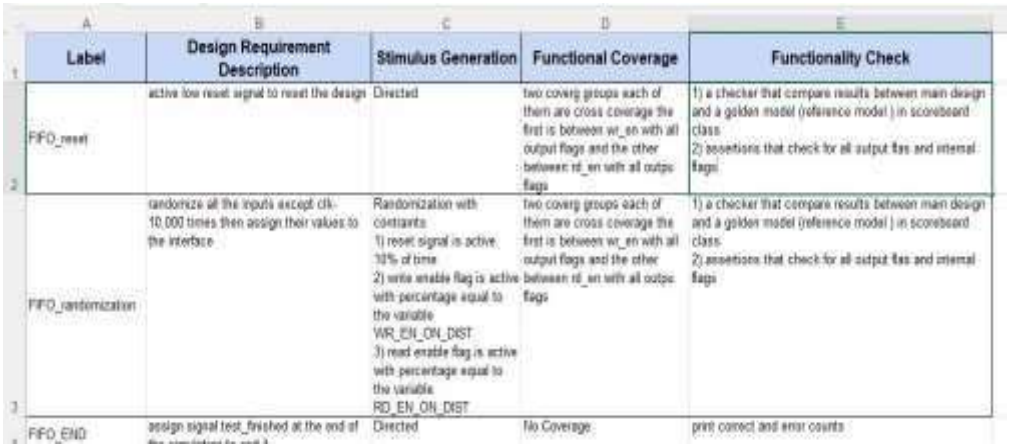
```
        bin seq_item_cov_almostempty_0_1[0]          [illegible]       1     Covered
        bin seq_item_cov_almostempty_0_1[1]              518       1     Covered
    Coverpoint cvr_group::write_acknowledge_cover      100.0%     100     Covered
        covered/total bins:                                2       2
        missing/total bins:                                0       2
        % Hit:                                         100.0%     100
        bin wr_ack_0_1[0]                               6116       1     Covered
        bin wr_ack_0_1[1]                               4085       1     Covered
    Cross cvr_group::write_cross_cov                   100.0%     100     Covered
        covered/total bins:                                7       7
        missing/total bins:                                0       7
        % Hit:                                         100.0%     100
        bin wr_full                                    10201       1     Covered
        bin wr_empty                                   10201       1     Covered
        bin wr_seq_item_cov_almostempty                10201       1     Covered
        bin wr_seq_item_cov_almostfull                 10201       1     Covered
        bin wr_wr_ack                                  10201       1     Covered
        bin wr_seq_item_cov_overflow                   10201       1     Covered
        bin wr_under                                   10201       1     Covered
    Cross cvr_group::read_cross_cov                    100.0%     100     Covered
        covered/total bins:                                7       7
        missing/total bins:                                0       7
        % Hit:                                         100.0%     100
        bin rd_full                                    10201       1     Covered
        bin rd_empty                                   10201       1     Covered
        bin rd_seq_item_cov_almostempty                10201       1     Covered
        bin rd_seq_item_cov_almostfull                 10201       1     Covered
        bin rd_wr_ack                                  10201       1     Covered
        bin rd_seq_item_cov_overflow                   10201       1     Covered
        bin rd_under                                   10201       1     Covered
 CLASS fifo_coverage

TOTAL COVERGROUP COVERAGE: 100.0%  COVERGROUP TYPES: 1
```

Sequential coverage snippets:

/uvm_pkg::uvm_re…	Immediate	SVA	on	0	0
/uvm_pkg::uvm_re…	Immediate	SVA	on	0	0
/fifo_write_read_s…	Immediate	SVA	on	0	1
/fifo_read_only_se…	Immediate	SVA	on	0	1
/fifo_write_only_se…	Immediate	SVA	on	0	1
/top/DUT/fifo_sva_…	Immediate	SVA	on	0	1
/top/DUT/fifo_sva_…	Immediate	SVA	on	0	1
/top/DUT/fifo_sva_…	Immediate	SVA	on	0	1
/top/DUT/fifo_sva_…	Immediate	SVA	on	0	1
/top/DUT/fifo_sva_…	Immediate	SVA	on	0	1
/top/DUT/fifo_sva_…	Concurrent	SVA	on	0	1
/top/DUT/fifo_sva_…	Concurrent	SVA	on	0	1
/top/DUT/fifo_sva_…	Concurrent	SVA	on	0	1
/top/DUT/fifo_sva_…	Concurrent	SVA	on	0	1
/top/DUT/fifo_sva_…	Concurrent	SVA	on	0	1
/top/DUT/fifo_sva_…	Concurrent	SVA	on	0	1
/top/DUT/fifo_sva_…	Concurrent	SVA	on	0	1
/top/DUT/fifo_sva_…	Concurrent	SVA	on	0	1

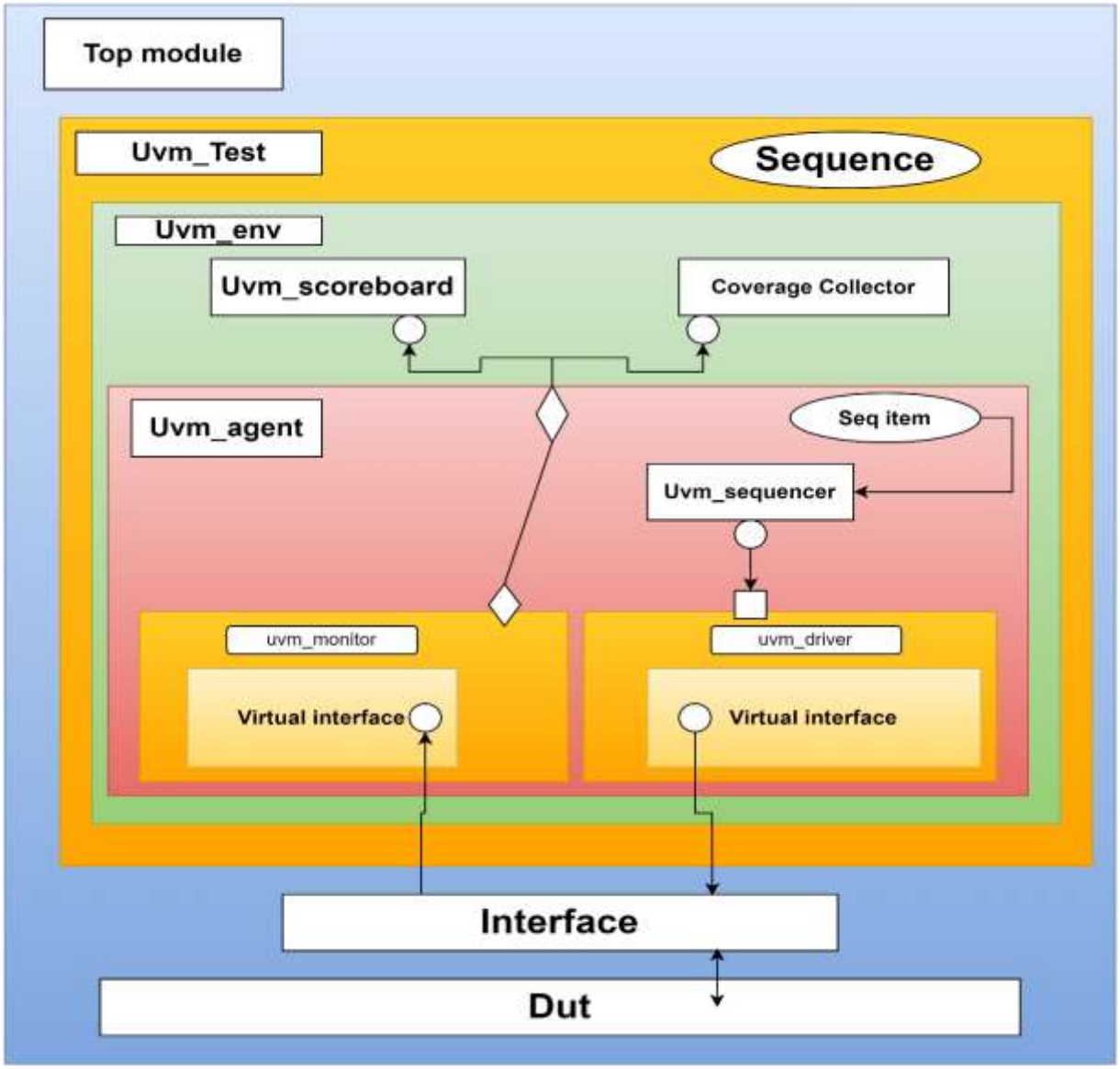
```
Name                    File(Line)            Failure Pass
                                              Count   Count
-------------------------------------------------------------
/top/DUT/fifo_sva_instance/assert__check_rd_ptr_increment
                        Fifo_assertions.sv(80)        0     1
/top/DUT/fifo_sva_instance/assert__check_wr_ptr_increment
                        Fifo_assertions.sv(72)        0     1
/top/DUT/fifo_sva_instance/assert__check_count_decrement
                        Fifo_assertions.sv(64)        0     1
/top/DUT/fifo_sva_instance/assert__check_count_increment
                        Fifo_assertions.sv(55)        0     1
/top/DUT/fifo_sva_instance/assert__check_count_stable
                        Fifo_assertions.sv(46)        0     1
/top/DUT/fifo_sva_instance/assert__check_underflow
                        Fifo_assertions.sv(38)        0     1
/top/DUT/fifo_sva_instance/assert__check_overflow
                        Fifo_assertions.sv(30)        0     1
/top/DUT/fifo_sva_instance/assert__check_wr_ack
                        Fifo_assertions.sv(22)        0     1
/top/DUT/fifo_sva_instance/reset
                        Fifo_assertions.sv(5)       0     1
/top/DUT/fifo_sva_instance/fulla
                        Fifo_assertions.sv(11)        0     1
/top/DUT/fifo_sva_instance/almostfulla
                        Fifo_assertions.sv(12)        0     1
/top/DUT/fifo_sva_instance/emptya
                        Fifo_assertions.sv(13)        0     1
/top/DUT/fifo_sva_instance/almostemptya
                        Fifo_assertions.sv(14)        0     1
/fifo_write_read_seq_pkg/fifo_write_read_seq/body/#ublk#164226423#15/immed__23
                        Fifo_write_read_seq.sv(23)        0     1
/fifo_read_only_seq_pkg/fifo_read_only_seq/body/#ublk#143532583#15/immed__25
                        Fifo_read_only_seq.sv(25)        0     1
/fifo_write_only_seq_pkg/fifo_write_only_seq/body/#ublk#220570231#15/immed__25
                        Fifo_write_only_seq.sv(25)        0     1
```

Verification plan snippet:



Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
FIFO_reset	active low reset signal to reset the design	Directed	two coverg groups each of them are cross coverage the first is between wr_en with all output flags and the other between rd_en with all outpu flags	1) a checker that compare results between main design and a golden model (reference model) in scoreboard class 2) assertions that check for all output flas and internal flags
FIFO_randomization	randomize all the inputs except clk- 10,000 times then assign their values to the interface	Randomization with constants 1) reset signal is active 10% of time 2) write enable flag is active with percentage equal to the variable WR_EN_ON_DIST 3) read enable flag is active with percentage equal to the variable RD_EN_ON_DIST	two coverg groups each of them are cross coverage the first is between wr_en with all output flags and the other between rd_en with all outpu flags	1) a checker that compare results between main design and a golden model (reference model) in scoreboard class 2) assertions that check for all output flas and internal flags
FIFO_END	assign signal test_finished at the end of	Directed	No Coverage	print correct and error counts

Uvm structure:



1. Top Module

The top module is where the UVM testbench is instantiated. It includes:

- The **DUT** , which is the hardware design being verified.
- The **interface** connects the DUT to the UVM environment and provides the required signals and their corresponding virtual interfaces for the driver and monitor components.

2. UVM_Test

The **UVM_Test** is the test class, which controls the simulation and stimulus generation process. It initializes and configures the environment, sequences, and components.This test runs a sequence of operations to drive input signals and monitor outputs.

Key responsibilities of **uvm_test** include:

- Instantiating the UVM environment (UVM_env).
- Running sequences that control the flow of transactions and data.
- Configuring stimulus, coverage, and any test-specific settings.

3. UVM_Env

The **UVM Environment (UVM_Env)** is where all verification components are brought together. It houses the agents, scoreboard, and coverage collectors. This environment is a container for all components needed to drive and monitor the DUT.

The environment is responsible for:

- Setting up communication between the agents and the scoreboard.
- Collecting coverage to ensure thorough verification.
- Instantiating the **UVM_Agent**, which coordinates the drivers, monitors, and sequencers.

4. UVM_Agent

The **UVM_Agent** acts as a container for the main components that interact with the DUT:

- **UVM_Sequencer**: This component is responsible for generating the sequence items (transactions) that are fed to the driver. The sequencer selects the next transaction from the sequence and passes it to the driver.
- **UVM_Driver**: This component converts the abstract sequence items into actual signal-level activity on the interface. It receives transactions from the sequencer and drives them onto the virtual interface connected to the DUT.
- **UVM_Monitor**: The monitor observes the interface activity, collects data (signals), and forwards the information to other components such as the scoreboard for analysis. It doesn't drive any signals but only observes them.

Sequence Flow in UVM_Agent:

- The sequencer generates a **sequence item**, which represents a transaction.
- The driver receives this item and converts it into actual signal activity on the interface, interacting with the DUT.
- The monitor captures the signal activity on the interface (both inputs and outputs), which is used for further analysis.

5. Interface

The **interface** is the physical connection between the UVM testbench and the DUT. It contains the signals required for communication, such as clock, reset, data, and control signals. The interface connects directly to the driver, which controls these signals during simulation, and to the monitor, which observes them.

Virtual interfaces are used within the driver and monitor to enable them to interact with the signals in the interface. These virtual interfaces provide a pointer to the actual hardware signals in the interface.

6. DUT

The **DUT** is the hardware design being verified. It is connected to the interface and responds to the signals driven by the testbench. The outputs of the DUT are observed and analyzed to verify its functionality.

7. UVM_Monitor

The **monitor** is a passive component that observes transactions on the interface without driving any signals. It captures all relevant data, which includes inputs sent to the DUT and outputs generated by the DUT. The monitor packages this data and sends it to other components for analysis:

- **UVM_Scoreboard** for functional checking.
- **Coverage Collector** for coverage analysis to ensure all test cases have been exercised.

8. UVM_Scoreboard

The **UVM_Scoreboard** checks the correctness of the DUT's behavior by comparing the observed outputs with the expected outputs. The scoreboard typically receives data from the monitor, compares it with a reference model or the expected results, and reports errors.

It serves two main purposes:

- **Functional Verification:** By comparing the expected and actual results.
- **Error Logging:** If any diffrences between expected and actual behavior are found, the scoreboard logs an error, helping identify bugs in the DUT.

9. Coverage Collector

The **Coverage Collector** gathers data on how well the testbench exercises the DUT. It checks for:

- Code coverage: how much of the design's code has been executed.
- Functional coverage: how many functional scenarios have been tested.
- Transaction coverage: how many different transactions or states have been exercised.

10. Sequence and Sequence Item

- The **sequence** generates and controls the flow of **sequence items** (transactions) from the sequencer to the driver. Sequence items are data packets or operations that are executed in the DUT.
- The **sequence item** contains the specific data or instructions to be applied to the DUT via the driver.

End-to-End Flow in UVM Testbench:

1. **(UVM_Test):** The top-level UVM_Test initiates the UVM environment and defines the verification plan.
2. **(UVM_Sequencer):** The sequencer generates sequence items to stimulate the DUT.
3. **(UVM_Driver):** The driver translates the abstract transactions into real signal activity on the DUT interface.
4. **(UVM_Monitor):** The monitor observes the activity on the interface, capturing both the input sent to the DUT and the output generated by it.
5. **(UVM_Scoreboard):** The scoreboard compares the actual outputs of the DUT with the expected results, checking for functional correctness.
6. **(Coverage Collector):** The coverage collector tracks which parts of the DUT and which test cases have been exercised to ensure thorough verification.