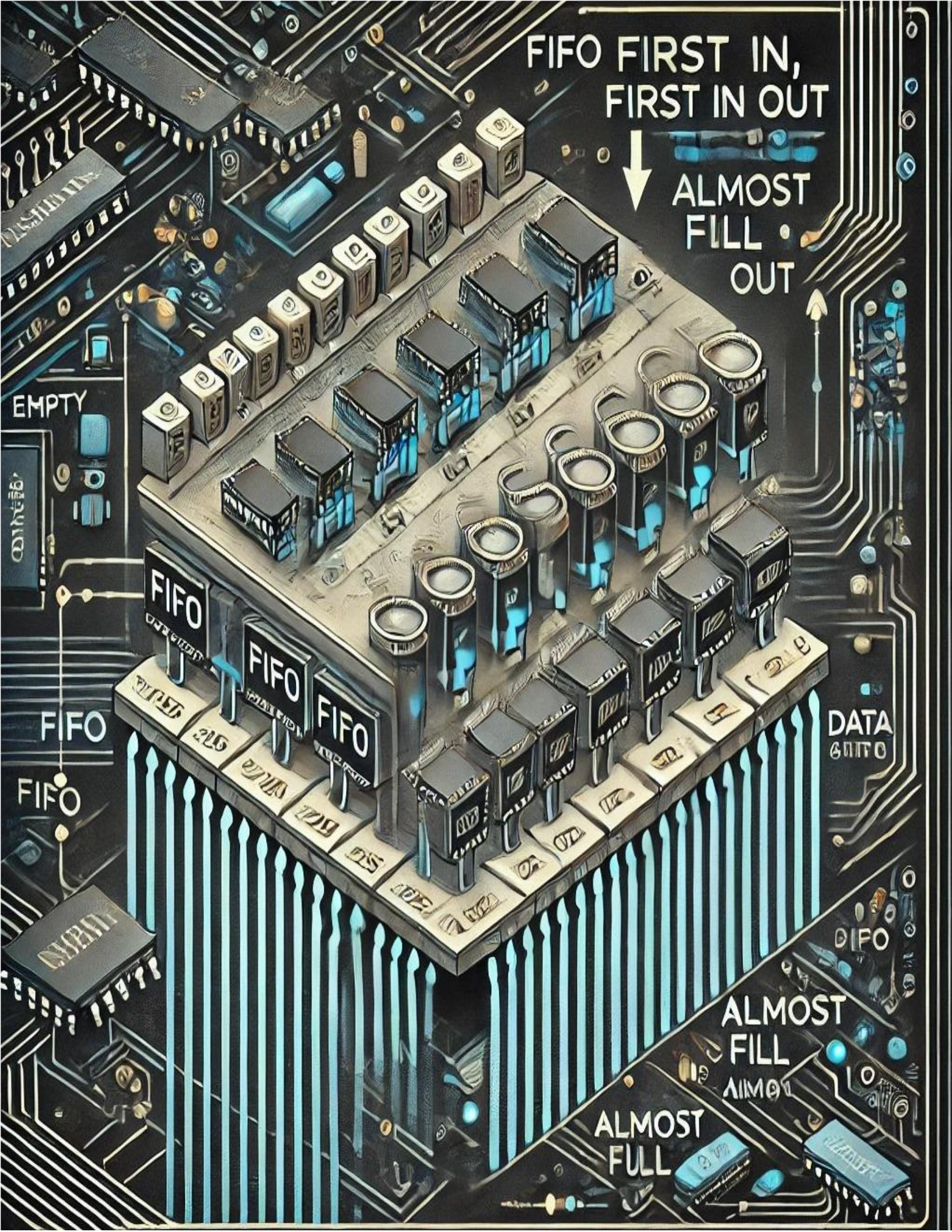

FIFO FIRST IN,
FIRST IN OUT
ALMOST
FILL
OUT
EMPTY
FIFO
FIFO
FIFO
FIFO
FIFO
DATA
ALMOST
FILL
ALMOST
FULL

FIFO Design and Verification using UVM

Introduction

The FIFO (First In, First Out) memory system is a type of data structure used in computer science and digital systems to manage data flow. The core principle of FIFO is that the first piece of data entering the memory is the first to exist. This is comparable to a queue in real life, such as a line at a checkout counter, where the first person in line is served first.

In a FIFO memory system, data is stored in a sequential manner, where the first element is always at the front, and new data is added to the end. It is commonly used in buffering, especially in systems that require synchronization between different data rates, such as communication protocols, audio/video streaming, or data pipelines.

Key Benefits of FIFO Memory:

1. **Efficient Data Flow Management:** FIFO ensures smooth data handling in systems where timing and order are critical, like networking, where packets must be processed in the order they are received.
2. **Simplicity and Predictability:** FIFO's structure is simple and easy to implement, which makes it ideal for managing first-come, first-served operations.
3. **Synchronization Across Different Data Speeds**: It is commonly used in systems where data producers and consumers operate at different speeds, ensuring that data is processed in the correct order without loss or overlap.
4. **Widespread Application:** FIFO memory systems are widely used in hardware and software design, including areas like microcontroller programming, memory management, and data streaming.

FIFO Architecture

Inputs:

- clk (Clock):
    - The clock signal used to synchronize the FIFO's operations (read and write).
    - The module uses this signal to drive the timing of internal processes.
- rst_n (Reset, active low):
    - A reset signal that initializes or resets the internal states (e.g., pointers, count) when rst_n is low.
- wr_en (Write Enable):
    - This signal controls whether data is written into the FIFO. If wr_en is high and the FIFO is not full, the data is written to the FIFO's memory.
- rd_en (Read Enable):
    - This signal controls whether data is read from the FIFO. If rd_en is high and the FIFO is not empty, data is read from the FIFO.
- data_in (Input Data):
    - The data being written into the FIFO when wr_en is high. It is a bit vector of size FIFO_WIDTH.

Outputs:

- data_out (Output Data):
    - The data that is read from the FIFO when rd_en is high. The data comes from the memory location indicated by the rd_ptr (read pointer).
- wr_ack (Write Acknowledge):
    - A signal that indicates whether a write operation has been successfully completed. If wr_en is high and data is written, wr_ack is set to 1.

- overflow:
    - This signal is set when a write operation is attempted but the FIFO is full. It indicates an overflow condition, meaning data has been lost due to insufficient space.
- underflow:
    - This signal is set when a read operation is attempted but the FIFO is empty. It indicates an underflow condition, meaning there is no data to read.
- full:
    - This signal goes high when the FIFO has reached its maximum capacity and can no longer accept new data for writing.
- empty:
    - This signal goes high when the FIFO has no data available for reading.
- almostfull:
    - This signal indicates that the FIFO is close to being full (one element away from being full). It helps manage the flow of data before the FIFO reaches full capacity.
- almostempty:
    - This signal indicates that the FIFO is close to being empty (only one element left). It helps manage the flow of data before the FIFO runs out of data.

FIFO Internal Signals & Logic

- **Internal Registers and Logic**

1. mem (Memory Array):
    - This is the actual memory used by the FIFO to store data. It has a depth of FIFO_DEPTH and each element has a width of FIFO_WIDTH.
    - Example: reg [FIFO_WIDTH-1:0] mem [FIFO_DEPTH-1:0];
2. wr_ptr (Write Pointer):
    - This pointer keeps track of the memory location where the next write operation should occur. It increments with every successful write and wraps around if it reaches the maximum depth of the FIFO.
3. rd_ptr (Read Pointer):
    - This pointer keeps track of the memory location from which the next read operation should take place. It increments with every successful read and wraps around when it reaches the maximum depth of the FIFO.
4. count (Data Count):
    - This register keeps track of the number of data elements stored in the FIFO. It increases when data is written and decrements when data is read.
    - count helps determine the status signals like full, empty, almostfull, and almostempty.

- **FIFO Control Logic**

- Write Operation:
    - Data is written into the FIFO when the wr_en signal is high and the FIFO is not full (count < FIFO_DEPTH).
    - The wr_ptr increments after every write operation to point to the next location in memory.
    - wr_ack is set high to acknowledge a successful write.

    - If a write is attempted while the FIFO is full, the overflow signal is set high.
- Read Operation:
    - Data is read from the FIFO when the rd_en signal is high and the FIFO is not empty (count > 0).
    - The rd_ptr increments after every read operation to point to the next data location.
    - If a read is attempted while the FIFO is empty, the underflow signal is set high.

Count Logic:

- The count is incremented when data is written and decremented when data is read.
- When both read and write operations are enabled simultaneously:
    - If the FIFO is not full, the count is increased.
    - If the FIFO is not empty, the count is decremented.
- This count value helps in controlling the signals:
    - full = (count == FIFO_DEPTH)
    - empty = (count == 0)
    - almostfull = (count == FIFO_DEPTH-1)
    - almostempty = (count == 1)

Summary of Signal Roles

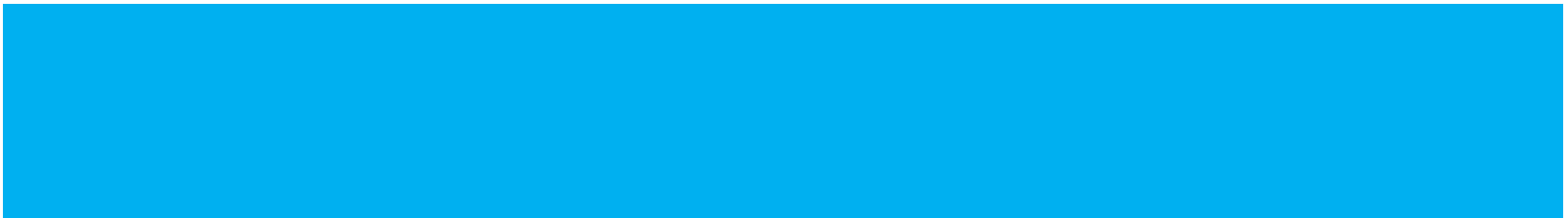
Signal Name	Type	Description
clk	*Input*	*Clock signal to synchronize the FIFO operations.*
rst_n	*Input*	*Active-low reset to initialize/reset FIFO pointers and control signals.*
data_in	*Input*	*Data to be written into the FIFO.*
wr_en	*Input*	*Enables writing operation when high*
rd_en	*Input*	*Enables read operation when high.*
data_out	*Output*	*Data being read from the FIFO.*
wr_ack	*Output*	*Acknowledges a successful writing operation.*
overflow	*Output*	*Indicates FIFO overflow when a write is attempted on a full FIFO.*
underflow	*Output*	*Indicates FIFO underflow when a read is attempted on an empty FIFO.*
full	*Output*	*Indicates the FIFO is full (no more data can be written).*
empty	*Output*	*Indicates the FIFO is empty (no more data is available to read).*
almostfull	*Output*	*Indicates the FIFO is almost full (one element away from full).*
almostempty	*Output*	*Indicates the FIFO is almost empty (one element left).*
wr_ptr	*Register*	*Tracks the next memory location for the write operation.*
rd_ptr	*Register*	*Tracks the next memory location for the read operation.*
count	*Register*	*Tracks the number of elements currently stored in the FIFO.*

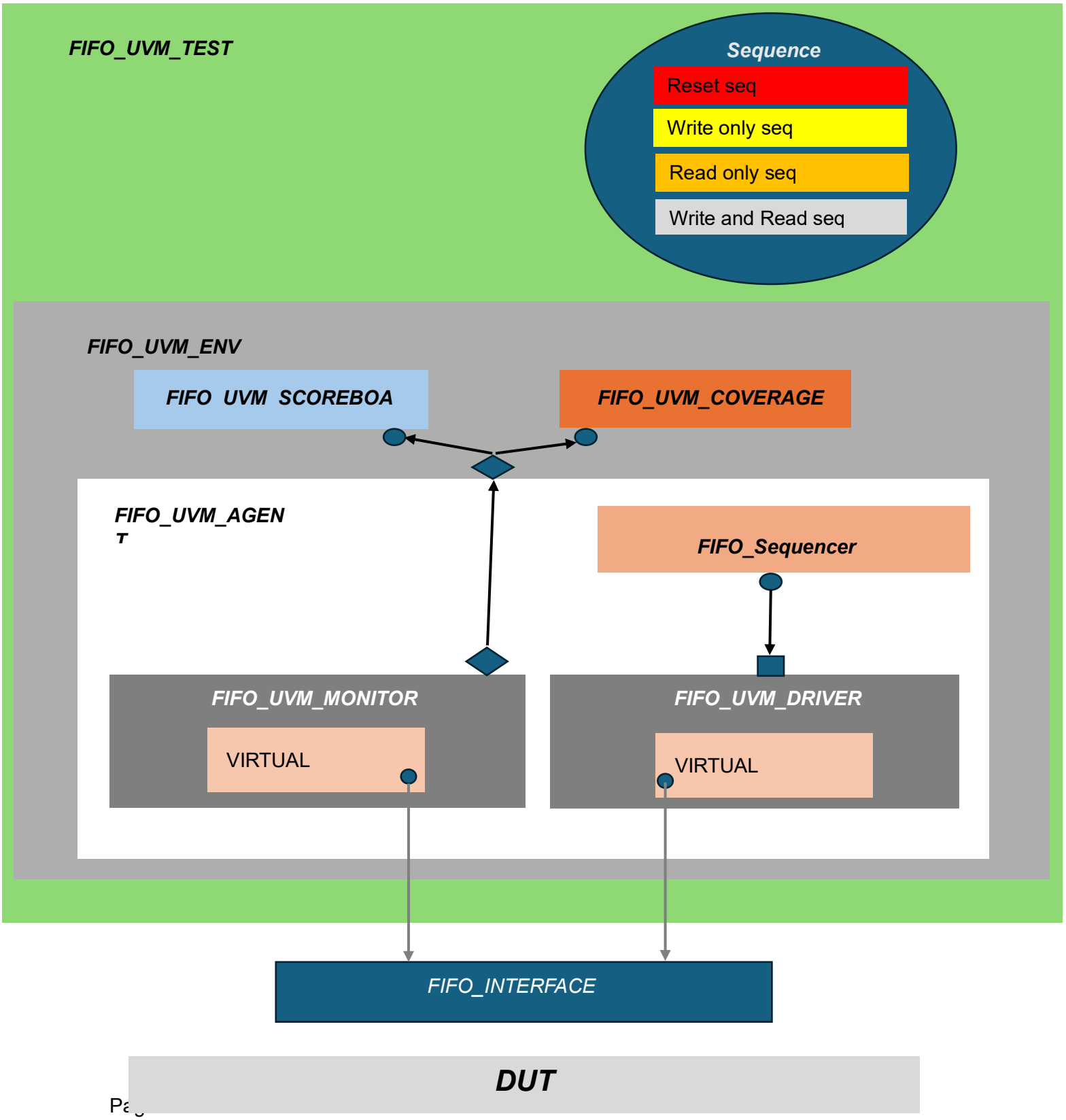
Verification Plan

Label	Description	Stimulus Generation	Functional Coverage	Functionality Check

FIFO_1	When the reset is asserted, the read pointer, write pointer, count overflow, underflow, write acknowlage are LOW	Directed at the start of the simulation	cover the reset signal	A Golden model in the scoreboar to make sure the output is correct and an immediate assertion also
FIFO_2	When writing enables asserted write ptr and counter increase by 1, and when read enable asserted read ptr and counter decreased by 1 , when both asserted the couter does't chang	Randomize during the simulation	-	Concurrent assertions to check the counter
FIFO_3	write only sequence	constraing writing only and no read during simulation	cover read and write enable and the output signals with them	A Golden model to check wr_ack ,almostfull , full , overflow and concurrent and immediate asserion also to check functionality
FIFO_4	read only sequence	constraing reading only and no write during simulation	cover read and write enable and the output signals with them	A Golden model to check almostempty , empty , underflow, data_out and concurrent and immediate asserion also to check functionality
FIFO_5	write and read sequence , at first write most , then read most , then the write and read are equal	constraing reading and writing during simulation	cover read and write enable and the output signals with them	A Golden model to check the outputs signal and concurrent and immediate asserion also to check functionality
FIFO_6	when empty , almostempty , full , almostfull		-	An immediate assertion to check them and A Golden model in the scoreboar to make sure they are correct
FIFO_7	data in and reset, write and read enable ,the data out asserted when readenable is high	Randomize during the simulation with constrain to reset to be more of the time off		A Golden model in the scoreboar to make sure the output is correct
FIFO_8	when writing after the fifo full overflow appear, when reading after fifo is empty underflow appear			A Golden model in the scoreboar to make sure the output is correct and a concurrent assertion also
FIFO_9			The coverage needed is cross coverage between 3 signals which are write enable, read enable and each output signal	

UVM_TESTBECH_STRUCTURAL




FIFO_TOP_MODU
FIFO_UVM_TEST
Sequence
Reset seq
Write only seq
Read only seq
Write and Read seq
FIFO_UVM_ENV
FIFO UVM SCOREBOA
FIFO_UVM_COVERAGE
FIFO_UVM_AGEN
T
FIFO_Sequencer
FIFO_UVM_MONITOR
VIRTUAL
FIFO_UVM_DRIVER
VIRTUAL
FIFO_INTERFACE
DUT

How UVM Testbench works

1. FIFO Top Module

The top module is responsible for instantiating the FIFO design under test (**DUT**) and configuring the testbench environment. It sets up the FIFO interface in the UVM configuration database, allowing access to the interface from the test module. The top module also generates the clock and reset signals necessary for the simulation.

2. FIFO Sequence Item

The sequence item defines the basic stimulus that will be applied to the DUT. It includes the randomized or constrained stimulus fields for the FIFO signals (such as data, read/write signals). Sequence items are used to send and receive data between the testbench and the FIFO design.

3. FIFO Sequence

The sequence controls how the sequence items are generated and in what order they are applied to the DUT. Different sequences can be created to validate various functionalities of the FIFO, such as:

- Reset sequence
- Write-only sequence
- Read-only sequence
- Combined read-write sequence

Each sequence ensures the correct behavior of the FIFO in different operational modes.

4. FIFO Configuration Object

The configuration object stores essential handles such as the interface to the DUT. The test module retrieves this interface from the configuration database, and the configuration object makes it available to the UVM components (like agents and drivers). This ensures that all components can interact correctly with the DUT interface.

5. FIFO Environment

The environment creates the agent, scoreboard, and coverage, and connects the agent to the scoreboard and coverage. This ensures the data from the monitor is sent to the scoreboard for checking and to the coverage for tracking what has been tested

6. FIFO Test

The test class is the main control center of the UVM testbench. It creates the environment and triggers the required sequences. It retrieves the virtual interface from the configuration database and sets up the configuration object. The test manages the flow of the simulation, including which sequence to execute, and sends sequence items to the sequencer. Essentially, the test orchestrates the entire verification process, making decisions about which operations to perform and when.

7. FIFO Driver

The driver is responsible for driving the stimulus from the sequencer to the DUT. When the test starts a sequence, the driver fetches sequence items from the sequencer and drives the corresponding signals on the DUT interface at the appropriate times. The driver directly interacts with the FIFO design, ensuring that the correct inputs are applied as per the test plan.

8. FIFO Monitor

The monitor observes the signals on the DUT interface and extracts relevant data for analysis. It captures the output from the FIFO design and passes this information to the agent, which in turn connects it to the coverage and scoreboard components for checking and validation. The monitor operates passively, without influencing the simulation.

9. FIFO Agent

The agent is responsible for creating and managing UVM components like the driver, monitor, and sequencer. It establishes the connections between the driver and the sequencer, as well as between the driver and the interface. Similarly, it connects the monitor to the interface and the agent. The agent ensures that all components work together cohesively during the simulation.

10. FIFO Scoreboard

The scoreboard is used for functional checking of the DUT. It contains a reference model, which simulates the expected behavior of the FIFO design. During the simulation, the scoreboard compares the actual output of the FIFO against this reference model, flagging any mismatches or errors. The scoreboard also tracks statistics, such as the number of mismatches and their specific details.

11. FIFO Coverage

The coverage component ensures that all functional aspects of the FIFO design are thoroughly verified. It contains covergroups and coverpoints that capture key functional scenarios. The coverage component samples these points during the simulation to ensure that all important features and edge cases of the FIFO are exercised.

Bugs in the design

First bug: resetting is not complete

```
24    always @(posedge clk or negedge rst_n) begin
25        if (!rst_n) begin
26            wr_ptr <= 0;
27        end
28        else if (wr_en && count < FIFO_DEPTH) begin
29            mem[wr_ptr] <= data_in;
```

Reset the other signals

```
always @(posedge FIFO_if.clk or negedge FIFO_if.rst_n) begin
    if (!FIFO_if.rst_n) begin
        rd_ptr <= 0; FIFO_if.underflow <=0; // reset underflow
    end
always @(posedge FIFO_if.clk or negedge FIFO_if.rst_n) begin
    if (!FIFO_if.rst_n) begin
        wr_ptr <= 0; FIFO_if.overflow <= 0; //  reseting overflow
        FIFO_if.wr_ack <=0 ; // reseting write acknowlage
    end
```

Second bug: Not handling overflow if read and write enable are high at the same time

```
else begin
    wr_ack <= 0;
    if (full & wr_en)
        overflow <= 1;
    else
        overflow <= 0;
end
```

Handling this bug

```
else begin
    FIFO_if.wr_ack <= 0;
    if (FIFO_if.full & FIFO_if.wr_en & !FIFO_if.rd_en)  // add codition if read and write are high overflow must = 0
        FIFO_if.overflow <= 1;
    else
        FIFO_if.overflow <= 0;
end
```

Third bug: underflow is sequential not combinational

```
66    assign underflow = (empty && rd_en)? 1 : 0;
```

Handling this bug

```
else begin
    if(FIFO_if.empty & !FIFO_if.wr_en & FIFO_if.rd_en) // underflow is sequential not combinational
        FIFO_if.underflow <=1 ;
    else
        FIFO_if.underflow <=0 ;
end
```

Fourth bug: the counter doesn't handle case of full or empty and write and read enables = 1

```
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;
    end
```

Handling this problem

```
always @(posedge FIFO_if.clk or negedge FIFO_if.rst_n) begin
    if (!FIFO_if.rst_n) begin
        count <= 0;
    end
    else begin // handling case if write and read enable = 1 in case of full and empty
        if  ( (({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b10) && !FIFO_if.full) || (FIFO_if.empty && ({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b11) ))
            count <= count + 1;
        else if ( (({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b01) && !FIFO_if.empty) || (FIFO_if.full && ({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b11) ) )
            count <= count - 1;
    end
end
```

Fifth bug: almost full doesn't take a right value

```
67    assign almostfull = (count == FIFO_DEPTH-2)? 1 : 0;
```

Handling this problem

```
66    assign FIFO_if.almostfull = (count == FIFO_if.FIFO_DEPTH-1)? 1 : 0; // almostfull at count = DEPTH-1
```

FIFO Design after editing bugs

```
module FIFO(FIFO_interface.DUT FIFO_if);

    // Calculate the maximum address size for the FIFO based on its depth
    localparam max_fifo_addr = $clog2(FIFO_if.FIFO_DEPTH);

    // Memory array to store FIFO data
    reg [FIFO_if.FIFO_WIDTH-1:0] mem [FIFO_if.FIFO_DEPTH-1:0];

    // Write and Read pointers
    reg [max_fifo_addr-1:0] wr_ptr, rd_ptr;
    // Count of elements in the FIFO
    reg [max_fifo_addr:0] count;

    // Write Operation
    always @(posedge FIFO_if.clk or negedge FIFO_if.rst_n) begin
        if (!FIFO_if.rst_n) begin
            wr_ptr <= 0;
            FIFO_if.overflow <= 0; // Reset overflow condition
            FIFO_if.wr_ack <= 0; // Reset write acknowledgment
        end
        else if (FIFO_if.wr_en && count < FIFO_if.FIFO_DEPTH) begin
            // Write data to memory and acknowledge
            mem[wr_ptr] <= FIFO_if.data_in;
            FIFO_if.wr_ack <= 1;
            wr_ptr <= wr_ptr + 1; // Increment write pointer
        end
        else begin
            FIFO_if.wr_ack <= 0;
            // Check for overflow condition when trying to write into a full FIFO
            if (FIFO_if.full && FIFO_if.wr_en && !FIFO_if.rd_en)
                FIFO_if.overflow <= 1;
            else
                FIFO_if.overflow <= 0; // Reset overflow if not full
        end
    end

    // Read Operation
    always @(posedge FIFO_if.clk or negedge FIFO_if.rst_n) begin
        if (!FIFO_if.rst_n) begin
            rd_ptr <= 0;
            FIFO_if.underflow <= 0; // Reset underflow condition
        end
        else if (FIFO_if.rd_en && count != 0) begin
```

```
            // Read data from memory
            FIFO_if.data_out <= mem[rd_ptr];
            rd_ptr <= rd_ptr + 1; // Increment read pointer
        end
        else begin
            // Check for underflow condition when trying to read from an empty
FIFO
            if (FIFO_if.empty && !FIFO_if.wr_en && FIFO_if.rd_en)
                FIFO_if.underflow <= 1;
            else
                FIFO_if.underflow <= 0; // Reset underflow if not empty
        end
    end

    // Count Logic
    always @(posedge FIFO_if.clk or negedge FIFO_if.rst_n) begin
        if (!FIFO_if.rst_n) begin
            count <= 0; // Initialize count on reset
        end
        else begin
            // Update count based on read/write enable signals and FIFO status
            if  ((({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b10) && !FIFO_if.full) ||
                  (FIFO_if.empty && ({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b11)))
                count <= count + 1; // Increment count for write

            else if ((({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b01) &&
!FIFO_if.empty) ||
                     (FIFO_if.full && ({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b11)))
                count <= count - 1; // Decrement count for read
        end
    end

    // Status Signals
    assign FIFO_if.full = (count == FIFO_if.FIFO_DEPTH) ? 1 : 0;
    assign FIFO_if.empty = (count == 0) ? 1 : 0;
    assign FIFO_if.almostfull = (count == FIFO_if.FIFO_DEPTH-1) ? 1 : 0; //
Almost full condition
    assign FIFO_if.almostempty = (count == 1) ? 1 : 0; // Almost empty condition

endmodule
```

FIFO Top module

```
import fifo_test_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
module FIFO_TOP ();

bit clk ;
initial begin
    clk = 1 ;
    forever begin
        #1 clk =~clk ;
    end
end
FIFO_interface FIFO_if (clk);
FIFO dut (FIFO_if) ;
bind FIFO fifo_assertions fifo_assertions_block (FIFO_if) ;
initial begin
    uvm_config_db # (virtual FIFO_interface ) :: set (null , "uvm_test_top",
"FIFO" , FIFO_if );
    run_test("fifo_test") ;
end

endmodule
```

FIFO Interface

```
interface FIFO_interface(clk) ;
input clk ;
parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;
bit [FIFO_WIDTH-1:0] data_in;
bit clk, rst_n, wr_en, rd_en;
logic [FIFO_WIDTH-1:0] data_out;
logic wr_ack, overflow, underflow;
logic  full, empty, almostfull, almostempty;

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

endinterface //interfacename
```

Assertions table

Feature	Assertion
Whenever FIFO is almost full the counter = fifo_depth -1	@(posedge clk) (almostfull |-> (count == FIFO_DEPTH-1))
Whenever FIFO is empty the counter = 0	@(posedge clk) (empty |-> (count == 0))
Whenever FIFO is almost empty counter = 1	@(posedge clk) (almostempty |-> (count == 1))
Reset conditions	@(posedge clk) (!rst_n |-> (wr_ptr == 0 && rd_ptr == 0 && wr_ack == 0 && overflow == 0 && underflow == 0 && count == 0))
Write pointer condition	@(posedge clk) (wr_en && count < FIFO_DEPTH |-> (wr_ptr == $past(wr_ptr+1'b1))
Read pointer condition	@(posedge clk) (rd_en && count != 0 |-> (rd_ptr == $past(rd_ptr+1'b1))
Counter up condition	@(posedge clk) ((({wr_en, rd_en} == 2'b10) && !full) || (empty && ({wr_en, rd_en} == 2'b11)) |-> (count == $past(count + 1'b1))
Counter down condition	@(posedge clk) ((({wr_en, rd_en} == 2'b01) && !empty) || (full && ({wr_en, rd_en} == 2'b11)) |-> (count == $past(count - 1'b1))
Write acknowledge condition	@(posedge clk) (wr_en && count < FIFO_DEPTH |-> (wr_ack == 1)
High overflow condition	@(posedge clk) (full & wr_en & !rd_en |-> (overflow == 1)
Low overflow condition	@(posedge clk) !(full & wr_en & !rd_en) |-> (overflow == 0)
High underflow condition	@(posedge clk) (empty & !wr_en & rd_en |-> (underflow == 1)
Low underflow condition	@(posedge clk) !(empty & !wr_en & rd_en) |-> (underflow == 0)

FIFO Assertion module

```
module fifo_assertions (FIFO_interface.DUT FIFO_if);

  // Assertion checks based on the FIFO count
  always_comb begin
    // Check for FULL condition
    if(dut.count == FIFO_if.FIFO_DEPTH)
      FULL: assert final (FIFO_if.full == 1);
    else
      NOT_FULL: assert final (FIFO_if.full == 0);

    // Check for ALMOSTFULL condition
    if(dut.count == FIFO_if.FIFO_DEPTH - 1)
      ALMOSTFULL: assert final (FIFO_if.almostfull == 1);
    else
      NOT_ALMOSTFULL: assert final (FIFO_if.almostfull == 0);

    // Check for EMPTY condition
    if(dut.count == 0)
      EMPTY: assert final (FIFO_if.empty == 1);
    else
      NOT_EMPTY: assert final (FIFO_if.empty == 0);

    // Check for ALMOSTEMPTY condition
    if(dut.count == 1)
      ALMOSTEMPTY: assert final (FIFO_if.almostempty == 1);
    else
      NOT_ALMOSTEMPTY: assert final (FIFO_if.almostempty == 0);

    // Reset condition checks
    if (!FIFO_if.rst_n) begin
      assert_reset: assert final (
        dut.wr_ptr == 0 &&
        dut.rd_ptr == 0 &&
        FIFO_if.wr_ack == 0 &&
        FIFO_if.overflow == 0 &&
        FIFO_if.underflow == 0 &&
        dut.count == 0
      );
    end
  end

  // Property for write pointer assertion
  property write_ptr;
```

```
  @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n)
    (FIFO_if.wr_en && dut.count < FIFO_if.FIFO_DEPTH) |=>
    (dut.wr_ptr == $past(dut.wr_ptr + 1'b1));
endproperty

// Property for read pointer assertion
property read_ptr;
  @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n)
    (FIFO_if.rd_en && dut.count != 0) |=>
    (dut.rd_ptr == $past(dut.rd_ptr + 1'b1));
endproperty

// Property for counting up
property counter_up;
  @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n)
    ((({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b10) && !FIFO_if.full) ||
    (FIFO_if.empty && ({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b11))) |=>
    (dut.count == $past(dut.count + 1'b1));
endproperty

// Property for counting down
property counter_down;
  @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n)
    ((({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b01) && !FIFO_if.empty) ||
    (FIFO_if.full && ({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b11))) |=>
    (dut.count == $past(dut.count - 1'b1));
endproperty

// Property for write acknowledge
property wr_acknowledge;
  @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n)
    (FIFO_if.wr_en && dut.count < FIFO_if.FIFO_DEPTH) |=>
    (FIFO_if.wr_ack == 1);
endproperty

// Property for high overflow condition
property high_overflow;
  @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n)
    (FIFO_if.full & FIFO_if.wr_en & !FIFO_if.rd_en) |=>
    (FIFO_if.overflow == 1);
endproperty

// Property for low overflow condition
property low_overflow;
  @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n)
```

```
      !(FIFO_if.full & FIFO_if.wr_en & !FIFO_if.rd_en) |=>
      (FIFO_if.overflow == 0);
  endproperty

  // Property for high underflow condition
  property high_underflow;
    @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n)
      (FIFO_if.empty & !FIFO_if.wr_en & FIFO_if.rd_en) |=>
      (FIFO_if.underflow == 1);
  endproperty

  // Property for low underflow condition
  property low_underflow;
    @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n)
      !(FIFO_if.empty & !FIFO_if.wr_en & FIFO_if.rd_en) |=>
      (FIFO_if.underflow == 0);
  endproperty

  // Assertions based on properties
  Write_pointer: assert property (write_ptr);
  Read_pointer: assert property (read_ptr);
  Counter_up: assert property (counter_up);
  Counter_down: assert property (counter_down);
  Write_acknowledge: assert property (wr_acknowledge);
  High_overflow: assert property (high_overflow);
  Low_overflow: assert property (low_overflow);
  High_underflow: assert property (high_underflow);
  Low_underflow: assert property (low_underflow);

  // Coverage assertions for properties
  Write_pointer_cover: cover property (write_ptr);
  Read_pointer_cover: cover property (read_ptr);
  Counter_up_cover: cover property (counter_up);
  Counter_down_cover: cover property (counter_down);
  Write_acknowledge_cover: cover property (wr_acknowledge);
  High_overflow_cover: cover property (high_overflow);
  Low_overflow_cover: cover property (low_overflow);
  High_underflow_cover: cover property (high_underflow);
  Low_underflow_cover: cover property (low_underflow);

endmodule
```

shared package

```
package shared_pkg ;
    int error_count  ;
    int correct_count  ;

endpackage
```

FIFO Sequence item

```
package fifo_seq_item_pkg ;
import uvm_pkg::*;
`include "uvm_macros.svh"
parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;
class fifo_seq_item extends uvm_sequence_item ;
`uvm_object_utils (fifo_seq_item)
// declare variables
int RD_EN_ON_DIST = 30 ;
int WR_EN_ON_DIST = 70 ;
rand logic [FIFO_WIDTH-1:0] data_in;
rand logic rst_n, wr_en, rd_en;
logic [FIFO_WIDTH-1:0] data_out;
logic wr_ack, overflow;
logic full, empty, almostfull, almostempty, underflow;
function new(string name = "fifo_seq_item");
    super.new(name);
endfunction // new
//  ------------constraint_randamization----------------------
constraint reset {
    rst_n dist {1:/99 , 0:/1} ; // active low reset most of the time
}
constraint Write_en {
    wr_en dist {1:/WR_EN_ON_DIST , 0:/ (100-WR_EN_ON_DIST) };
}
constraint Read_en {
    rd_en dist {1:/RD_EN_ON_DIST , 0:/(100-RD_EN_ON_DIST)};
}

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

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

endclass //className

endpackage
```

FIFO Reset Sequence

```
package fifo_reset_seq_pkg ;
    import fifo_seq_item_pkg::*;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class fifo_reset_seq extends uvm_sequence #(fifo_seq_item);
    `uvm_object_utils(fifo_reset_seq)
    fifo_seq_item seq_item ;

        function new(string name = "fifo_reset_seq" );
            super.new(name) ;
        endfunction //new()
        task body ;
            seq_item = fifo_seq_item :: type_id :: create("seq_item");
            start_item (seq_item) ;
            assert(seq_item.randomize()) ;
            seq_item.rst_n = 0 ;
            finish_item(seq_item) ;
        endtask
    endclass //fifo_reset_seq extends superClass
endpackage
```

FIFO Write only Sequence

```
package fifo_write_only_seq_pkg ;
    import fifo_seq_item_pkg::*;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class fifo_write_only_seq extends uvm_sequence #(fifo_seq_item);
    `uvm_object_utils(fifo_write_only_seq)
    fifo_seq_item seq_item ;
    int read_dist = 0 ; // No read
    int write_dist = 100 ; // Write only
        function new(string name = "fifo_write_only_seq" );
            super.new(name) ;
        endfunction //new()
        task body ;
           repeat (50) begin
            seq_item = fifo_seq_item :: type_id :: create("seq_item");
            seq_item.RD_EN_ON_DIST = read_dist;
            seq_item.WR_EN_ON_DIST = write_dist;
            start_item (seq_item) ;
            assert(seq_item.randomize()) ;
            finish_item(seq_item) ;
           end
        endtask //automatic
    endclass //fifo_write_only_seq extends superClass
endpackage
```

FIFO Read only sequence

```
package fifo_read_only_seq_pkg ;
    import fifo_seq_item_pkg::*;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class fifo_read_only_seq extends uvm_sequence #(fifo_seq_item);
    `uvm_object_utils(fifo_read_only_seq)
    fifo_seq_item seq_item ;
    int read_dist = 100 ; // Read only
    int write_dist = 0 ; // No write
        function new(string name = "fifo_read_only_seq" );
            super.new(name) ;
        endfunction //new()
        task body ;
           repeat (50) begin
            seq_item = fifo_seq_item :: type_id :: create("seq_item");
```

```
            seq_item.RD_EN_ON_DIST = read_dist;
            seq_item.WR_EN_ON_DIST = write_dist;
            start_item (seq_item) ;
            assert(seq_item.randomize()) ;
            finish_item(seq_item) ;
           end
        endtask //automatic
    endclass //fifo_read_only_seq extends superClass
endpackage
```

FIFO Write and Read Sequence

```
package fifo_write_read_seq_pkg ;
    import fifo_seq_item_pkg::*;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class fifo_write_read_seq extends uvm_sequence #(fifo_seq_item);
    `uvm_object_utils(fifo_write_read_seq)
    fifo_seq_item seq_item ;
    int read_dist  ;
    int write_dist ;

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

        task body ;
        // Randomize transactions with mostly writes
        read_dist = 30 ; write_dist = 70 ;
            repeat (500) begin
                seq_item = fifo_seq_item :: type_id :: create("seq_item");
                seq_item.RD_EN_ON_DIST = read_dist;
                seq_item.WR_EN_ON_DIST = write_dist;
                start_item (seq_item) ;
                assert(seq_item.randomize()) ;
                finish_item(seq_item) ;
            end

        // Randomize transactions with mostly writes
        read_dist = 70 ; write_dist = 50 ;
            repeat (500) begin
                seq_item = fifo_seq_item :: type_id :: create("seq_item");
                seq_item.RD_EN_ON_DIST = read_dist;
                seq_item.WR_EN_ON_DIST = write_dist;
```

```
                start_item (seq_item) ;
                assert(seq_item.randomize()) ;
                finish_item(seq_item) ;
            end

        // Randomize transactions with equal reads and writes
        read_dist = 50 ; write_dist = 50 ;
            repeat (500) begin
                seq_item = fifo_seq_item :: type_id :: create("seq_item");
                seq_item.RD_EN_ON_DIST = read_dist;
                seq_item.WR_EN_ON_DIST = write_dist;
                start_item (seq_item) ;
                assert(seq_item.randomize()) ;
                finish_item(seq_item) ;
            end

        endtask

    endclass //fifo_write_read_seq extends superClass
endpackage
```

FIFO Sequencer

```
package my_sequencer_pkg ;
import fifo_seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class my_sequencer extends uvm_sequencer #(fifo_seq_item);
`uvm_component_utils(my_sequencer)
    function new(string name = "my_sequencer" , uvm_component parent = null);
        super.new(name,parent) ;
    endfunction //new()
endclass //className extends superClass

endpackage
```

FIFO Configuration object

```
package fifo_config_obj_pkg ;
import uvm_pkg::*;
`include "uvm_macros.svh"
class fifo_config_obj extends uvm_object ;
`uvm_object_utils (fifo_config_obj)
virtual FIFO_interface fifo_vif ;
    function new(string name = "fifo_config_obj");
        super.new(name);
    endfunction //new()
endclass //className extends superClass
endpackage
```

FIFO Driver

```
package fifo_driver_pkg ;
import fifo_seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class fifo_driver extends uvm_driver #(fifo_seq_item);
`uvm_component_utils(fifo_driver)
virtual FIFO_interface fifo_vif ;
fifo_seq_item seq_item ;
    function new(string name = "fifo_driver" , uvm_component parent = null);
        super.new(name,parent);
    endfunction //new()
task  run_phase (uvm_phase phase );
    super.run_phase(phase) ;

    forever begin
        seq_item = fifo_seq_item :: type_id :: create ("seq_item");
        seq_item_port.get_next_item(seq_item);
        fifo_vif.data_in = seq_item.data_in ;
        fifo_vif.rst_n   = seq_item.rst_n ;
        fifo_vif.wr_en   = seq_item.wr_en ;
        fifo_vif.rd_en   = seq_item.rd_en ;
        @(negedge fifo_vif.clk);
        seq_item_port.item_done() ;
        `uvm_info("run_phase" , seq_item.convert2string_stimulus() , UVM_HIGH)
    end

endtask

endclass //className extends superClass
endpackage
```

FIFO Monitor

```
package fifo_monitor_pkg ;
import uvm_pkg::*;
import fifo_seq_item_pkg::*;
`include "uvm_macros.svh"
class fifo_monitor extends uvm_monitor;
`uvm_component_utils(fifo_monitor)
virtual FIFO_interface fifo_vif ;
fifo_seq_item seq_item;
uvm_analysis_port #(fifo_seq_item) mon_ap ; // monitor analysis port
    function new(string name = "fifo_monitor" , uvm_component parent = null );
        super.new(name,parent) ;
    endfunction //new()

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

    task  run_phase(uvm_phase phase );
        super.run_phase(phase) ;
        forever begin
            seq_item = fifo_seq_item :: type_id :: create("seq_item") ;
            @(negedge fifo_vif.clk);
            seq_item.data_in     = fifo_vif.data_in;
            seq_item.rst_n       = fifo_vif.rst_n;
            seq_item.wr_en       = fifo_vif.wr_en;
            seq_item.rd_en       = fifo_vif.rd_en;
            seq_item.wr_ack      = fifo_vif.wr_ack;
            seq_item.overflow    = fifo_vif.overflow;
            seq_item.underflow   = fifo_vif.underflow;
            seq_item.full        = fifo_vif.full;
            seq_item.empty       = fifo_vif.empty;
            seq_item.almostfull  = fifo_vif.almostfull;
            seq_item.almostempty = fifo_vif.almostempty;
            seq_item.data_out    = fifo_vif.data_out;

            mon_ap.write(seq_item) ;
            `uvm_info ("run_phase", seq_item.convert2string(),UVM_HIGH)
        end
    endtask //automatic
endclass //className extends superClass

endpackage
```

FIFO Agent

```
package fifo_agent_pkg ;
import fifo_seq_item_pkg::*;
import my_sequencer_pkg::*;
import fifo_driver_pkg::*;
import fifo_monitor_pkg::*;
import fifo_config_obj_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class fifo_agent extends uvm_agent ;
`uvm_component_utils (fifo_agent)

my_sequencer sqr ;
fifo_driver driver ;
fifo_monitor monitor ;
fifo_config_obj cfg_obj ;
uvm_analysis_port #(fifo_seq_item) agt_ap ;

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

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        if(!uvm_config_db #(fifo_config_obj) :: get (this, "" ,"FIFOVIF",
cfg_obj))
        `uvm_fatal("build phase" , "Agent - unable to get configration object ")
        driver = fifo_driver :: type_id :: create ("driver", this);
        sqr = my_sequencer :: type_id :: create("sqr",this) ;
        monitor = fifo_monitor :: type_id :: create ("monitor", this) ;
        agt_ap = new("agt_ap" ,this) ;
    endfunction

    function void connect_phase(uvm_phase phase );
        driver.fifo_vif  = cfg_obj.fifo_vif;
        monitor.fifo_vif = cfg_obj.fifo_vif;
        driver.seq_item_port.connect(sqr.seq_item_export);
        monitor.mon_ap.connect(agt_ap);
    endfunction

endclass //className extends superClass
endpackage
```

FIFO Scoreboard

```
package fifo_scoreboard_pkg ;
    import shared_pkg::*;
    import fifo_seq_item_pkg::*;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    class fifo_scoreboard extends uvm_scoreboard ;
        `uvm_component_utils(fifo_scoreboard)
        uvm_analysis_export #(fifo_seq_item) sb_export ;
        uvm_tlm_analysis_fifo #(fifo_seq_item) sb_fifo ;
        fifo_seq_item seq_item_sb ;

        const int FULL_SIZE = FIFO_DEPTH ;
        const int ALMOST_FULL_SIZE = FIFO_DEPTH-1 ;
        const int EMPTY_SIZE = 0 ;
        const int ALMOST_EMPTY_SIZE = 1 ;
        int actual_size = 0 ; // the actual size of the fifo after reading and
writing

        logic[FIFO_WIDTH-1:0] fifo_ref [$] ;
        logic[FIFO_WIDTH-1:0] data_out_ref;
        logic wr_ack_ref, overflow_ref ;
        logic full_ref, empty_ref, almostfull_ref, almostempty_ref,
underflow_ref;

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

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

        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            sb_export.connect(sb_fifo.analysis_export) ;
        endfunction

        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
```

```
                sb_fifo.get(seq_item_sb) ;
                reference_model(seq_item_sb) ;
                if (seq_item_sb.data_out != data_out_ref) begin
                    $display("ERROR data_out_observed  = 0h%0h ,
data_out_expected = 0h%0h ", seq_item_sb.data_out , data_out_ref);
                    error_count++ ;
                end
                else begin
                    correct_count++ ;
                end

                // extra part checking other outputs
                if (seq_item_sb.full != full_ref || seq_item_sb.almostfull !=
almostfull_ref || seq_item_sb.empty != empty_ref
                || seq_item_sb.almostempty != almostempty_ref ||
seq_item_sb.wr_ack != wr_ack_ref || seq_item_sb.overflow != overflow_ref
                || seq_item_sb.underflow != underflow_ref ) begin
                $display ("Error in flags ") ;
                error_count++ ;
                end
            end
        endtask

        task reference_model(fifo_seq_item seq_item_check );

                if (!seq_item_check.rst_n) begin
                    fifo_ref.delete();
                    overflow_ref = 0 ; underflow_ref = 0 ;
                    actual_size = 0 ; wr_ack_ref = 0 ;
                end
                else begin
                    // seting default values for sequential outputs except
data_out to be assigned with it until no condition change it
                    overflow_ref = 0 ; underflow_ref = 0 ;
                    // push and pop data from the fifo
                    case ({seq_item_check.wr_en , seq_item_check.rd_en})
                        2'b11: begin // write and read
                            if(actual_size != FULL_SIZE && actual_size !=
EMPTY_SIZE) begin
                                fifo_ref.push_back(seq_item_check.data_in) ;
                                data_out_ref = fifo_ref.pop_front() ;
                                wr_ack_ref = 1 ;
                            end
                            else if (actual_size == FULL_SIZE) begin
                                data_out_ref = fifo_ref.pop_front();
```

```
                            actual_size = actual_size - 1 ;
                            wr_ack_ref = 0 ;
                        end
                        else if (actual_size == EMPTY_SIZE) begin
                            fifo_ref.push_back(seq_item_check.data_in) ;
                            actual_size = actual_size+1 ;
                            wr_ack_ref = 1 ;
                        end
                    end
                    2'b10 : begin // write only
                        if(actual_size != FULL_SIZE) begin
                            fifo_ref.push_back (seq_item_check.data_in);
                            actual_size = actual_size + 1 ;
                            wr_ack_ref = 1 ;
                        end
                        else begin
                            overflow_ref = 1 ; wr_ack_ref =  0 ;
                        end
                    end
                    2'b01 : begin // read only
                        if(actual_size != EMPTY_SIZE) begin
                            data_out_ref = fifo_ref.pop_front() ;
                            actual_size = actual_size - 1 ;
                            wr_ack_ref = 0 ;
                        end
                        else begin
                            underflow_ref = 1 ;
                            wr_ack_ref = 0 ;
                        end
                    end
                    2'b00 : begin
                        wr_ack_ref = 0 ; overflow_ref = 0 ; underflow_ref =
0;
                    end
                endcase
            end
            almostempty_ref = 0 ; empty_ref = 0 ;
            almostfull_ref = 0 ; full_ref = 0 ;
            if (actual_size == FULL_SIZE) begin
                full_ref = 1 ;
            end
            else if (actual_size == ALMOST_FULL_SIZE) begin
                almostfull_ref = 1 ;
            end
            else if (actual_size == EMPTY_SIZE) begin
```

```
                        empty_ref = 1 ;
                    end
                    else if (actual_size == ALMOST_EMPTY_SIZE) begin
                        almostempty_ref = 1 ;
                    end
        endtask

        function void report_phase(uvm_phase phase);
            super.report_phase(phase);
            `uvm_info("report_phase", $sformatf("Total successful transactions:
%0d", correct_count), UVM_MEDIUM);
            `uvm_info("report_phase", $sformatf("Total failed transactions: %0d",
error_count), UVM_MEDIUM);
        endfunction

    endclass
endpackage
```

FIFO Coverage

```
package fifo_coverage_pkg ;
import fifo_seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class fifo_coverage extends uvm_component ;
`uvm_component_utils(fifo_coverage)
uvm_analysis_export #(fifo_seq_item) cov_export;
uvm_tlm_analysis_fifo #(fifo_seq_item) cov_fifo;
fifo_seq_item seq_item_cov ;

covergroup  cvr_gp ;
Reset : coverpoint seq_item_cov.rst_n ;
Read_EN : coverpoint seq_item_cov.rd_en ;
Write_EN : coverpoint seq_item_cov.wr_en ;
Write_acknowledge : coverpoint seq_item_cov.wr_ack ;
Overflow : coverpoint seq_item_cov.overflow ;
FULL : coverpoint seq_item_cov.full ;
EMPTY : coverpoint seq_item_cov.empty ;
ALMOSTFULL : coverpoint seq_item_cov.almostfull ;
ALMOSTEMPTY : coverpoint seq_item_cov.almostempty ;
UNDERFLOW : coverpoint seq_item_cov.underflow ;
// crossing read_en and write_en with the outputs
Cross_rd_wr_writer_ack : cross Read_EN , Write_EN , Write_acknowledge {
```

```
    illegal_bins high_wr_ack_low_write = binsof(Write_acknowledge) intersect {1}
&& binsof(Write_EN) intersect {0} ; // they mustn't happened
}
Cross_rd_wr_Overflow : cross Read_EN , Write_EN , Overflow {
    illegal_bins high_overflow_low_write = binsof(Overflow) intersect {1} &&
binsof(Write_EN) intersect {0} ; // they mustn't happened
    illegal_bins high_rd_full_low_write = binsof(Write_EN) intersect {1} &&
binsof (Read_EN) intersect {1} && binsof(Overflow) intersect {1} ; // they
mustn't happend
}
Cross_rd_wr_FULL : cross Read_EN , Write_EN , FULL {
    illegal_bins all_high = binsof(Write_EN) intersect {1} && binsof (Read_EN)
intersect {1} && binsof(FULL) intersect {1} ; // they mustn't happend
    illegal_bins high_rd_full_low_write = binsof(Write_EN) intersect {0} &&
binsof (Read_EN) intersect {1} && binsof(FULL) intersect {1} ; // they mustn't
happend
}
Cross_rd_wr_EMPTY : cross Read_EN , Write_EN , EMPTY ;
Cross_rd_wr_ALMOSTFULL : cross Read_EN , Write_EN , ALMOSTFULL ;
Cross_rd_wr_ALMOSTEMPTY : cross Read_EN , Write_EN , ALMOSTEMPTY ;
Cross_rd_wr_UNDERFLOW : cross Read_EN , Write_EN , UNDERFLOW {
    illegal_bins high_underflow_low_read = binsof(UNDERFLOW) intersect {1} &&
binsof(Read_EN) intersect {0} ; // they mustn't happened
    illegal_bins high_rd_full_low_write = binsof(Write_EN) intersect {1} &&
binsof (Read_EN) intersect {1} && binsof(UNDERFLOW) intersect {1} ; // they
mustn't happend
}
endgroup

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

    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(seq_item_cov);
            cvr_gp.sample();
        end
    endtask


endclass //className
endpackage
```

FIFO 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 agent ;
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);
        agent = fifo_agent :: type_id :: create ("agent",this);
        sb = fifo_scoreboard :: type_id :: create ("sb",this);
        cov = fifo_coverage :: type_id :: create ("cov",this);
    endfunction : build_phase

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

FIFO Test

```
package fifo_test_pkg  ;
import fifo_env_pkg::*;
import fifo_reset_seq_pkg::*;
import fifo_write_only_seq_pkg::*;
import fifo_read_only_seq_pkg::*;
import fifo_write_read_seq_pkg::*;
import fifo_config_obj_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"

class fifo_test extends uvm_test;
`uvm_component_utils(fifo_test)
fifo_env env ;
fifo_reset_seq rst_seq ;
fifo_write_only_seq wr_seq ;
fifo_read_only_seq rd_seq ;
fifo_write_read_seq wr_rd_seq ;
fifo_config_obj cfg_obj_test ;
    function new(string name = "fifo_test" , uvm_component parent = null);
        super.new(name,parent);
    endfunction //new()
    function void build_phase (uvm_phase phase);
        super.build_phase(phase);
        env = fifo_env :: type_id :: create ("env" , this) ; // this refere to
this test class which is the parent for env
        cfg_obj_test = fifo_config_obj :: type_id :: create ("cfg_obj_test") ;
        rst_seq = fifo_reset_seq :: type_id :: create ("rst_seq") ;
        wr_seq = fifo_write_only_seq :: type_id :: create ("wr_seq") ;
        rd_seq = fifo_read_only_seq :: type_id :: create ("rd_seq") ;
        wr_rd_seq = fifo_write_read_seq :: type_id :: create ("wr_rd_seq") ;

        if(!uvm_config_db # (virtual FIFO_interface) :: get (this
,"","FIFO",cfg_obj_test.fifo_vif )) begin
            `uvm_fatal ("build_phase" , "TEST unable to get the virtual interface
of the FIFO form configration data base ") ;
        end
        uvm_config_db # (fifo_config_obj) :: set (this
,"*","FIFOVIF",cfg_obj_test ) ;
    endfunction

    task  run_phase(uvm_phase phase);
        super.run_phase (phase) ;
        phase.raise_objection (this) ;
```

```
        // reset sequence
        `uvm_info ("run_phase" , "Reseting Started ",UVM_LOW)
        rst_seq.start(env.agent.sqr) ;
        `uvm_info ("run_phase" , "Reseting  Ended ",UVM_LOW)

        // writing only sequence
         `uvm_info ("run_phase" , "Writing only Started ",UVM_LOW)
        wr_seq.start(env.agent.sqr) ;
        `uvm_info ("run_phase" , "Writing only Ended ",UVM_LOW)

        // Reading only sequence
        `uvm_info ("run_phase" , "Reading only Started ",UVM_LOW)
        rd_seq.start(env.agent.sqr) ;
        `uvm_info ("run_phase" , "Reading only Ended ",UVM_LOW)

        // Writing and Reading  sequence
        `uvm_info ("run_phase" , "Writing and Reading Started ",UVM_LOW)
        wr_rd_seq.start(env.agent.sqr) ;
        `uvm_info ("run_phase" , "Writing and Reading Ended ",UVM_LOW)

        phase.drop_objection(this) ;
    endtask : run_phase

endclass : fifo_test

endpackage
```

FIFO Source files

```
FIFO.sv
fifo_interface.sv
fifo_assertions.sv
fifo_seq_item_pkg.sv
fifo_reset_seq_pkg.sv
fifo_write_only_seq_pkg.sv
fifo_read_only_seq_pkg.sv
fifo_write_read_seq_pkg.sv
fifo_config_obj_pkg.sv
my_sequencer_pkg.sv
fifo_driver_pkg.sv
fifo_monitor_pkg.sv
fifo_agent_pkg.sv
fifo_scoreboard_pkg.sv
fifo_coverage_pkg.sv
fifo_env_pkg.sv
fifo_test_pkg.sv
FIFO_TOP.sv
```

FIFO Do file

```
vlib work
vlog -f src_files.list.txt +cover -covercells
vsim -voptargs=+acc work.FIFO_TOP -classdebug -uvmcontrol=all -
coverage -assertcover
add wave /FIFO_TOP/FIFO_if/*
coverage save FIFO.ucdb -onexit
run -all
```

Questa sim snippets

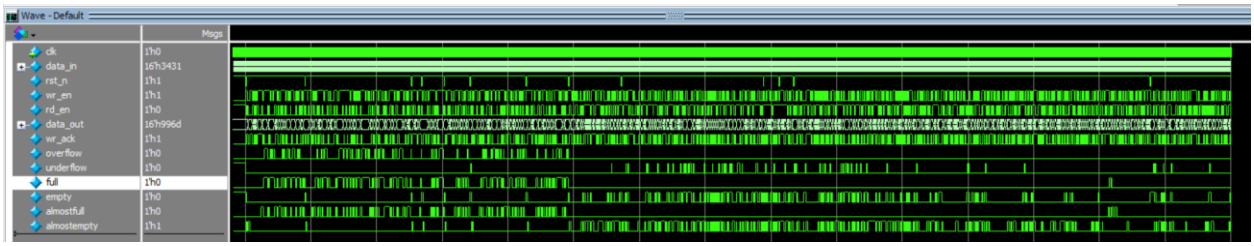


Figure 1: full wave

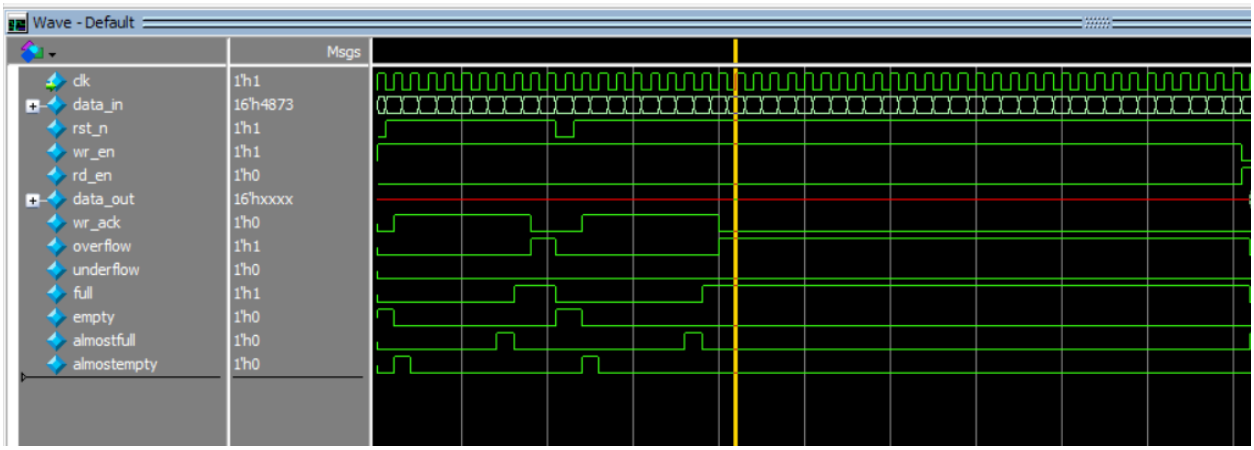


Figure 2: when writing only

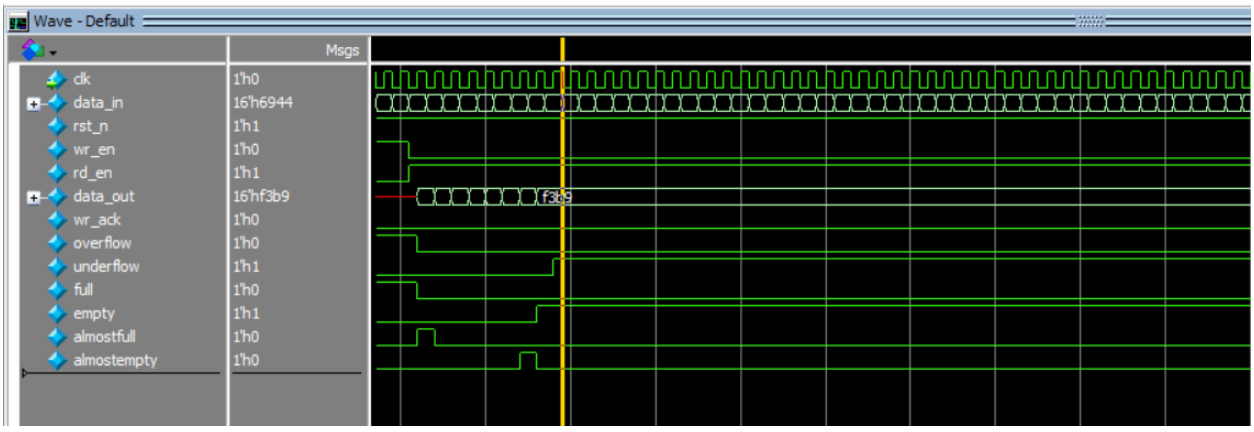


Figure 3: when reading only

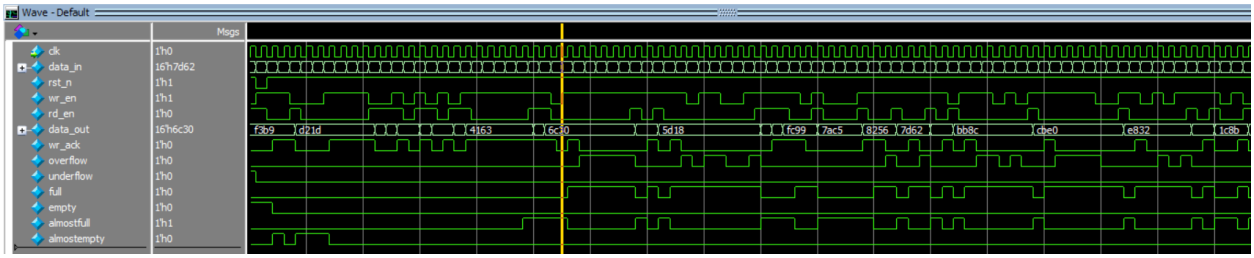


Figure 4:when most writes

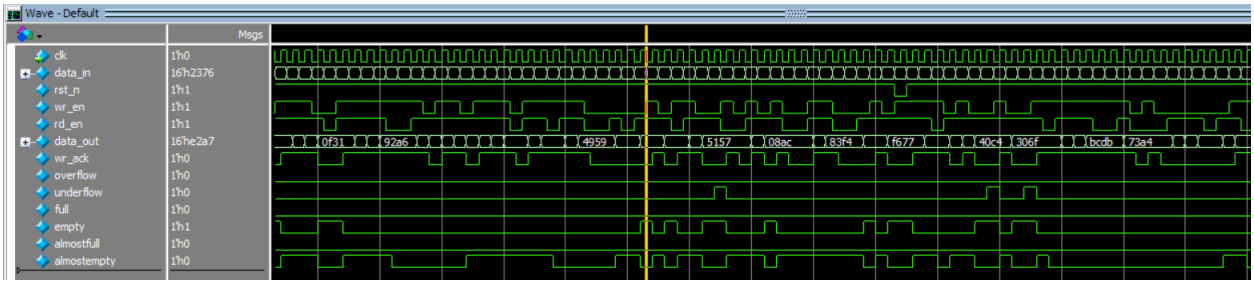


Figure 5: when most reads

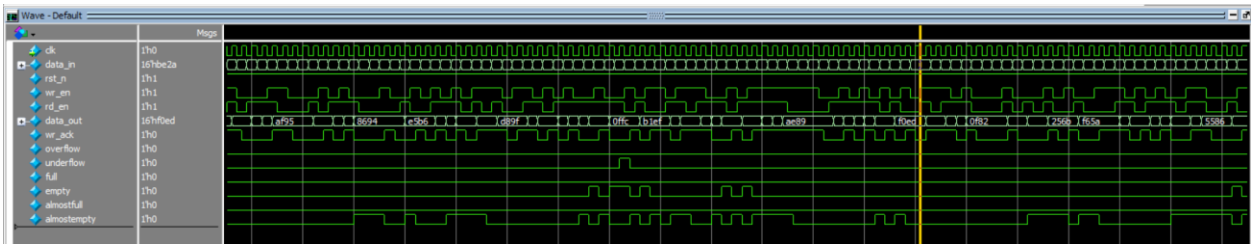
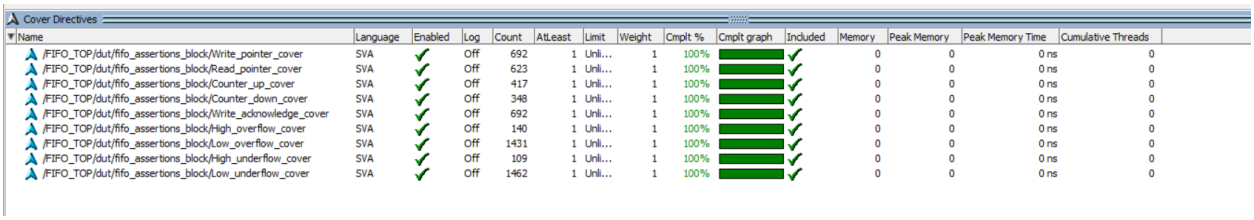


Figure 6: when equal read and write

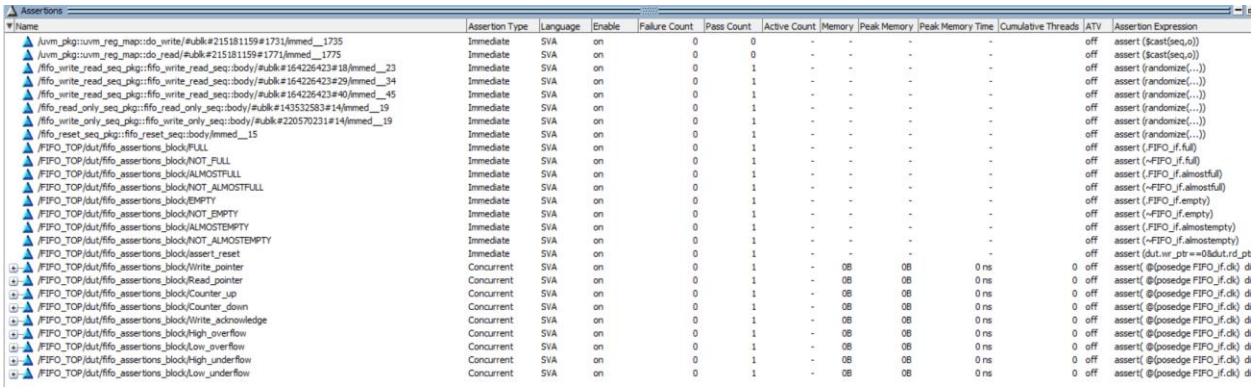
```
#   with `UVM_OBJECT_MUST_HAVE_CONSTRUCTOR undefined.
#   See http://www.eda.org/svdb/view.php?id=3770 for more details.
#
#       (Specify +UVM_NO_RELNOTES to turn off this notice)
#
# UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(277) @ 0: reporter [Questa UVM] QUESTA_UVM-1.2.3
# UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(278) @ 0: reporter [Questa UVM]  questa_uvm::init(+struct)
# UVM_INFO @ 0: reporter [RNTST] Running test fifo_test...
# UVM_INFO fifo_test_pkg.sv(41) @ 0: uvm_test_top [run_phase] Reseting Started
# UVM_INFO fifo_test_pkg.sv(43) @ 1: uvm_test_top [run_phase] Reseting  Ended
# UVM_INFO fifo_test_pkg.sv(46) @ 1: uvm_test_top [run_phase] Writing only Started
# UVM_INFO fifo_test_pkg.sv(48) @ 101: uvm_test_top [run_phase] Writing only Ended
# UVM_INFO fifo_test_pkg.sv(51) @ 101: uvm_test_top [run_phase] Reading only Started
# UVM_INFO fifo_test_pkg.sv(53) @ 201: uvm_test_top [run_phase] Reading only Ended
# UVM_INFO fifo_test_pkg.sv(56) @ 201: uvm_test_top [run_phase] Writing and Reading Started
# UVM_INFO fifo_test_pkg.sv(58) @ 3201: uvm_test_top [run_phase] Writing and Reading Ended
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 3201: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO fifo_scoreboard_pkg.sv(134) @ 3201: uvm_test_top.env.sb [report_phase] Total successful transactions: 1601
# UVM_INFO fifo_scoreboard_pkg.sv(135) @ 3201: uvm_test_top.env.sb [report_phase] Total failed transactions: 0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   14
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# [report_phase]     2
# [run_phase]     8
# ** Note: $finish    : C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 3201 ns  Iteration: 61  Instance: /FIFO_TOP
# 1
# Break in Task uvm_pkg/uvm_root::run_test at C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh line 430
```

Figure 7: UVM report


Cover Directives

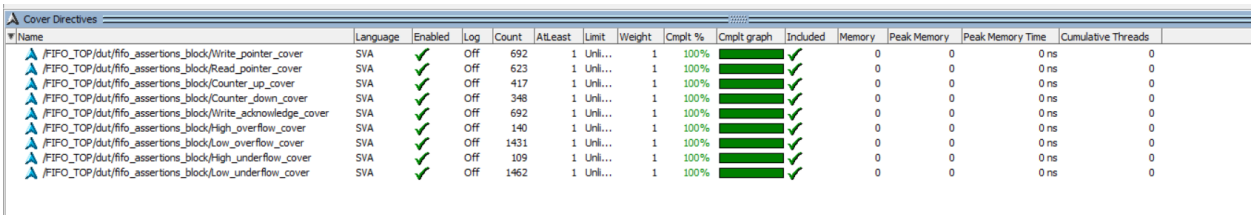
Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
/FIFO_TOP/dut/fifo_assertions_block/Write_pointer_cover	SVA	✓	Off	692	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_TOP/dut/fifo_assertions_block/Read_pointer_cover	SVA	✓	Off	623	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_TOP/dut/fifo_assertions_block/Counter_up_cover	SVA	✓	Off	417	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_TOP/dut/fifo_assertions_block/Counter_down_cover	SVA	✓	Off	348	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_TOP/dut/fifo_assertions_block/Write_acknowledge_cover	SVA	✓	Off	692	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_TOP/dut/fifo_assertions_block/High_overflow_cover	SVA	✓	Off	140	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_TOP/dut/fifo_assertions_block/Low_overflow_cover	SVA	✓	Off	1431	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_TOP/dut/fifo_assertions_block/High_underflow_cover	SVA	✓	Off	109	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_TOP/dut/fifo_assertions_block/Low_underflow_cover	SVA	✓	Off	1462	1	Unli...	1	100%		✓	0	0	0 ns	0

Figure 8: Assertion Coverage from Questa sim


Assertions

Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Count	Memory	Peak Memory	Peak Memory Time	Cumulative Threads	ATV	Assertion Expression
/uvm_pkg::uvm_reg_map::do_write/#ublk#215181159#1731/immed__1735	Immediate	SVA	on	0	0	-	-	-	-		off	assert ($cast(seq,o))
/uvm_pkg::uvm_reg_map::do_read/#ublk#215181159#1771/immed__1775	Immediate	SVA	on	0	0	-	-	-	-		off	assert ($cast(seq,o))
/fifo_write_read_seq_pkg::fifo_write_read_seq::body/#ublk#164226423#18/immed__23	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))
/fifo_write_read_seq_pkg::fifo_write_read_seq::body/#ublk#164226423#29/immed__34	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))
/fifo_write_read_seq_pkg::fifo_write_read_seq::body/#ublk#164226423#40/immed__45	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))
/fifo_read_only_seq_pkg::fifo_read_only_seq::body/#ublk#143532583#14/immed__19	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))
/fifo_write_only_seq_pkg::fifo_write_only_seq::body/#ublk#220570231#14/immed__19	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))
/fifo_reset_seq_pkg::fifo_reset_seq::body/immed__15	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))
/FIFO_TOP/dut/fifo_assertions_block/FULL	Immediate	SVA	on	0	1	-	-	-	-		off	assert (.FIFO_if.full)
/FIFO_TOP/dut/fifo_assertions_block/NOT_FULL	Immediate	SVA	on	0	1	-	-	-	-		off	assert (~FIFO_if.full)
/FIFO_TOP/dut/fifo_assertions_block/ALMOSTFULL	Immediate	SVA	on	0	1	-	-	-	-		off	assert (.FIFO_if.almostfull)
/FIFO_TOP/dut/fifo_assertions_block/NOT_ALMOSTFULL	Immediate	SVA	on	0	1	-	-	-	-		off	assert (~FIFO_if.almostfull)
/FIFO_TOP/dut/fifo_assertions_block/EMPTY	Immediate	SVA	on	0	1	-	-	-	-		off	assert (.FIFO_if.empty)
/FIFO_TOP/dut/fifo_assertions_block/NOT_EMPTY	Immediate	SVA	on	0	1	-	-	-	-		off	assert (~FIFO_if.empty)
/FIFO_TOP/dut/fifo_assertions_block/ALMOSTEMPTY	Immediate	SVA	on	0	1	-	-	-	-		off	assert (.FIFO_if.almostempty)
/FIFO_TOP/dut/fifo_assertions_block/NOT_ALMOSTEMPTY	Immediate	SVA	on	0	1	-	-	-	-		off	assert (~FIFO_if.almostempty)
/FIFO_TOP/dut/fifo_assertions_block/assert_reset	Immediate	SVA	on	0	1	-	-	-	-		off	assert (dut.wr_ptr==0&dut.rd_pt
/FIFO_TOP/dut/fifo_assertions_block/Write_pointer	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.clk) di
/FIFO_TOP/dut/fifo_assertions_block/Read_pointer	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.clk) di
/FIFO_TOP/dut/fifo_assertions_block/Counter_up	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.clk) di
/FIFO_TOP/dut/fifo_assertions_block/Counter_down	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.clk) di
/FIFO_TOP/dut/fifo_assertions_block/Write_acknowledge	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.clk) di
/FIFO_TOP/dut/fifo_assertions_block/High_overflow	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.clk) di
/FIFO_TOP/dut/fifo_assertions_block/Low_overflow	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.clk) di
/FIFO_TOP/dut/fifo_assertions_block/High_underflow	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.clk) di
/FIFO_TOP/dut/fifo_assertions_block/Low_underflow	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.clk) di

Figure 9 : Function Coverage from Questa sim


Cover Directives

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
/FIFO_TOP/dut/fifo_assertions_block/Write_pointer_cover	SVA	✓	Off	692	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_TOP/dut/fifo_assertions_block/Read_pointer_cover	SVA	✓	Off	623	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_TOP/dut/fifo_assertions_block/Counter_up_cover	SVA	✓	Off	417	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_TOP/dut/fifo_assertions_block/Counter_down_cover	SVA	✓	Off	348	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_TOP/dut/fifo_assertions_block/Write_acknowledge_cover	SVA	✓	Off	692	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_TOP/dut/fifo_assertions_block/High_overflow_cover	SVA	✓	Off	140	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_TOP/dut/fifo_assertions_block/Low_overflow_cover	SVA	✓	Off	1431	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_TOP/dut/fifo_assertions_block/High_underflow_cover	SVA	✓	Off	109	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_TOP/dut/fifo_assertions_block/Low_underflow_cover	SVA	✓	Off	1462	1	Unli...	1	100%		✓	0	0	0 ns	0

Figure 10: Directives from Questa sim

Coverage report

○ Assertion Coverage

```
================================================================================
=== Instance: /FIFO_TOP/dut/fifo_assertions_block
=== Design Unit: work.fifo_assertions
================================================================================

Assertion Coverage:
    Assertions                      18        18         0   100.00%
--------------------------------------------------------------------------------------------------------------------------------
Name                 File(Line)                    Failure      Pass     Vacuous     Disable     Attempt     Active Peak Active ATV
                                                   Count        Count    Count       Count       Count       Count  Count
--------------------------------------------------------------------------------------------------------------------------------
/FIFO_TOP/dut/fifo_assertions_block/FULL
                     fifo_assertions.sv(7)               0          1          0           0           1           0           - off
/FIFO_TOP/dut/fifo_assertions_block/NOT_FULL
                     fifo_assertions.sv(9)               0          1          0           0           1           0           - off
/FIFO_TOP/dut/fifo_assertions_block/ALMOSTFULL
                     fifo_assertions.sv(13)              0          1          0           0           1           0           - off
/FIFO_TOP/dut/fifo_assertions_block/NOT_ALMOSTFULL
                     fifo_assertions.sv(15)              0          1          0           0           1           0           - off
/FIFO_TOP/dut/fifo_assertions_block/EMPTY
                     fifo_assertions.sv(19)              0          1          0           0           1           0           - off
/FIFO_TOP/dut/fifo_assertions_block/NOT_EMPTY
                     fifo_assertions.sv(21)              0          1          0           0           1           0           - off
/FIFO_TOP/dut/fifo_assertions_block/ALMOSTEMPTY
                     fifo_assertions.sv(25)              0          1          0           0           1           0           - off
/FIFO_TOP/dut/fifo_assertions_block/NOT_ALMOSTEMPTY
                     fifo_assertions.sv(27)              0          1          0           0           1           0           - off
/FIFO_TOP/dut/fifo_assertions_block/assert_reset
                     fifo_assertions.sv(32)              0          1          0           0           1           0           - off
/FIFO_TOP/dut/fifo_assertions_block/Write_pointer
                     fifo_assertions.sv(108)             0        692        889          19        1601           1           - off
/FIFO_TOP/dut/fifo_assertions_block/Read_pointer
                     fifo_assertions.sv(109)             0        623        959          19        1601           0           - off
/FIFO_TOP/dut/fifo_assertions_block/Counter_up
                     fifo_assertions.sv(110)             0        417       1166          17        1601           1           - off
/FIFO_TOP/dut/fifo_assertions_block/Counter_down
                     fifo_assertions.sv(111)             0        348       1236          17        1601           0           - off
/FIFO_TOP/dut/fifo_assertions_block/Write_acknowledge
                     fifo_assertions.sv(112)             0        692        889          19        1601           1           - off
/FIFO_TOP/dut/fifo_assertions_block/High_overflow
                     fifo_assertions.sv(113)             0        140       1442          19        1601           0           - off
/FIFO_TOP/dut/fifo_assertions_block/Low_overflow
                     fifo_assertions.sv(114)             0       1431        144          25        1601           1           - off
/FIFO_TOP/dut/fifo_assertions_block/High_underflow
                     fifo_assertions.sv(115)             0        109       1476          16        1601           0           - off
/FIFO_TOP/dut/fifo_assertions_block/Low_underflow
                     fifo_assertions.sv(116)             0       1462        110          28        1601           1           - off
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        10        10         0   100.00%

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

○ Function Coverage

```
COVERGROUP COVERAGE:
----------------------------------------------------------------------------------------------------------
Covergroup                                                Metric       Goal       Bins    Status

----------------------------------------------------------------------------------------------------------
 TYPE /fifo_coverage_pkg/fifo_coverage/cvr_gp           100.00%        100          -    Covered
    covered/total bins:                                      66         66          -
    missing/total bins:                                       0         66          -
    % Hit:                                              100.00%        100          -
    Coverpoint Reset                                    100.00%        100          -    Covered
        covered/total bins:                                   2          2          -
        missing/total bins:                                   0          2          -
        % Hit:                                          100.00%        100          -
        bin auto[0]                                          15          1          -    Covered
        bin auto[1]                                        1586          1          -    Covered
    Coverpoint Read_EN                                  100.00%        100          -    Covered
        covered/total bins:                                   2          2          -
        missing/total bins:                                   0          2          -
        % Hit:                                          100.00%        100          -
        bin auto[0]                                         771          1          -    Covered
        bin auto[1]                                         830          1          -    Covered
    Coverpoint Write_EN                                 100.00%        100          -    Covered
        covered/total bins:                                   2          2          -
        missing/total bins:                                   0          2          -
        % Hit:                                          100.00%        100          -
        bin auto[0]                                         698          1          -    Covered
        bin auto[1]                                         903          1          -    Covered
    Coverpoint Write_acknowledge                        100.00%        100          -    Covered
        covered/total bins:                                   2          2          -
        missing/total bins:                                   0          2          -
        % Hit:                                          100.00%        100          -
        bin auto[0]                                         904          1          -    Covered
        bin auto[1]                                         697          1          -    Covered
    Coverpoint Overflow                                 100.00%        100          -    Covered
        covered/total bins:                                   2          2          -
        missing/total bins:                                   0          2          -
        % Hit:                                          100.00%        100          -
        bin auto[0]                                        1457          1          -    Covered
        bin auto[1]                                         144          1          -    Covered
    Coverpoint FULL                                     100.00%        100          -    Covered
        covered/total bins:                                   2          2          -
        missing/total bins:                                   0          2          -
        % Hit:                                          100.00%        100          -
        bin auto[0]                                        1336          1          -    Covered
        bin auto[1]                                         265          1          -    Covered
    Coverpoint EMPTY                                    100.00%        100          -    Covered
        covered/total bins:                                   2          2          -
        missing/total bins:                                   0          2          -
        % Hit:                                          100.00%        100          -
        bin auto[0]                                        1314          1          -    Covered
        bin auto[1]                                         287          1          -    Covered
    Coverpoint ALMOSTFULL                               100.00%        100          -    Covered
        covered/total bins:                                   2          2          -
        missing/total bins:                                   0          2          -
        % Hit:                                          100.00%        100          -
        bin auto[0]                                        1451          1          -    Covered
        bin auto[1]                                         150          1          -    Covered
    Coverpoint ALMOSTEMPTY                              100.00%        100          -    Covered
```

Coverpoint ALMOSTEMPTY	100.00%	100	-	Covered
covered/total bins:	2	2	-	
missing/total bins:	0	2	-	
% Hit:	100.00%	100	-	
bin auto[0]	1214	1	-	Covered
bin auto[1]	387	1	-	Covered
Coverpoint UNDERFLOW	100.00%	100	-	Covered
covered/total bins:	2	2	-	
missing/total bins:	0	2	-	
% Hit:	100.00%	100	-	
bin auto[0]	1491	1	-	Covered
bin auto[1]	110	1	-	Covered
Cross Cross_rd_wr_writer_ack	100.00%	100	-	Covered
covered/total bins:	6	6	-	
missing/total bins:	0	6	-	
% Hit:	100.00%	100	-	
Auto, Default and User Defined Bins:				
bin <auto[1],auto[1],auto[1]>	362	1	-	Covered
bin <auto[1],auto[1],auto[0]>	56	1	-	Covered
bin <auto[0],auto[1],auto[1]>	335	1	-	Covered
bin <auto[0],auto[1],auto[0]>	150	1	-	Covered
bin <auto[1],auto[0],auto[0]>	412	1	-	Covered
bin <auto[0],auto[0],auto[0]>	286	1	-	Covered
Illegal and Ignore Bins:				
illegal_bin high_wr_ack_low_write	0		-	ZERO
Cross Cross_rd_wr_Overflow	100.00%	100	-	Covered
covered/total bins:	5	5	-	
missing/total bins:	0	5	-	
% Hit:	100.00%	100	-	
Auto, Default and User Defined Bins:				
bin <auto[1],auto[1],auto[0]>	418	1	-	Covered
bin <auto[1],auto[0],auto[0]>	412	1	-	Covered
bin <auto[0],auto[1],auto[1]>	144	1	-	Covered
bin <auto[0],auto[1],auto[0]>	341	1	-	Covered
bin <auto[0],auto[0],auto[0]>	286	1	-	Covered
Illegal and Ignore Bins:				
illegal_bin high_rd_full_low_write	0		-	ZERO
illegal_bin high_overflow_low_write	0		-	ZERO
Cross Cross_rd_wr_FULL	100.00%	100	-	Covered
covered/total bins:	6	6	-	
missing/total bins:	0	6	-	
% Hit:	100.00%	100	-	
Auto, Default and User Defined Bins:				
bin <auto[0],auto[1],auto[1]>	221	1	-	Covered
bin <auto[0],auto[0],auto[1]>	44	1	-	Covered
bin <auto[1],auto[1],auto[0]>	418	1	-	Covered
bin <auto[1],auto[0],auto[0]>	412	1	-	Covered
bin <auto[0],auto[1],auto[0]>	264	1	-	Covered
bin <auto[0],auto[0],auto[0]>	242	1	-	Covered
Illegal and Ignore Bins:				
illegal_bin high_rd_full_low_write	0		-	ZERO
illegal_bin all_high	0		-	ZERO
Cross Cross_rd_wr_EMPTY	100.00%	100	-	Covered
covered/total bins:	8	8	-	
missing/total bins:	0	8	-	
% Hit:	100.00%	100	-	
Auto, Default and User Defined Bins:				
bin <auto[1],auto[1],auto[1]>	5	1	-	Covered
bin <auto[0],auto[1],auto[1]>	6	1	-	Covered
bin <auto[1],auto[0],auto[1]>	235	1	-	Covered

```
        Illegal and Ignore Bins:
            illegal_bin high_rd_full_low_write              0                    -    ZERO
            illegal_bin high_overflow_low_write             0                    -    ZERO
    Cross Cross_rd_wr_FULL                              100.00%        100           -    Covered
        covered/total bins:                                   6          6           -
        missing/total bins:                                   0          6           -
        % Hit:                                           100.00%        100           -
        Auto, Default and User Defined Bins:
            bin <auto[0],auto[1],auto[1]>                   221          1           -    Covered
            bin <auto[0],auto[0],auto[1]>                    44          1           -    Covered
            bin <auto[1],auto[1],auto[0]>                   418          1           -    Covered
            bin <auto[1],auto[0],auto[0]>                   412          1           -    Covered
            bin <auto[0],auto[1],auto[0]>                   264          1           -    Covered
            bin <auto[0],auto[0],auto[0]>                   242          1           -    Covered
        Illegal and Ignore Bins:
            illegal_bin high_rd_full_low_write              0                    -    ZERO
            illegal_bin all_high                            0                    -    ZERO
    Cross Cross_rd_wr_EMPTY                             100.00%        100           -    Covered
        covered/total bins:                                   8          8           -
        missing/total bins:                                   0          8           -
        % Hit:                                           100.00%        100           -
        Auto, Default and User Defined Bins:
            bin <auto[1],auto[1],auto[1]>                     5          1           -    Covered
            bin <auto[0],auto[1],auto[1]>                     6          1           -    Covered
            bin <auto[1],auto[0],auto[1]>                   235          1           -    Covered
            bin <auto[0],auto[0],auto[1]>                    41          1           -    Covered
            bin <auto[1],auto[1],auto[0]>                   413          1           -    Covered
            bin <auto[0],auto[1],auto[0]>                   479          1           -    Covered
            bin <auto[1],auto[0],auto[0]>                   177          1           -    Covered
            bin <auto[0],auto[0],auto[0]>                   245          1           -    Covered
    Cross Cross_rd_wr_ALMOSTFULL                        100.00%        100           -    Covered
        covered/total bins:                                   8          8           -
        missing/total bins:                                   0          8           -
        % Hit:                                           100.00%        100           -
        Auto, Default and User Defined Bins:
            bin <auto[1],auto[1],auto[1]>                    83          1           -    Covered
            bin <auto[0],auto[1],auto[1]>                    21          1           -    Covered
            bin <auto[1],auto[0],auto[1]>                    20          1           -    Covered
            bin <auto[0],auto[0],auto[1]>                    26          1           -    Covered
            bin <auto[1],auto[1],auto[0]>                   335          1           -    Covered
            bin <auto[0],auto[1],auto[0]>                   464          1           -    Covered
            bin <auto[1],auto[0],auto[0]>                   392          1           -    Covered
            bin <auto[0],auto[0],auto[0]>                   260          1           -    Covered
    Cross Cross_rd_wr_ALMOSTEMPTY                       100.00%        100           -    Covered
        covered/total bins:                                   8          8           -
        missing/total bins:                                   0          8           -
        % Hit:                                           100.00%        100           -
        Auto, Default and User Defined Bins:
            bin <auto[1],auto[1],auto[1]>                   208          1           -    Covered
            bin <auto[0],auto[1],auto[1]>                    50          1           -    Covered
            bin <auto[1],auto[0],auto[1]>                    59          1           -    Covered
            bin <auto[0],auto[0],auto[1]>                    70          1           -    Covered
            bin <auto[1],auto[1],auto[0]>                   210          1           -    Covered
            bin <auto[0],auto[1],auto[0]>                   435          1           -    Covered
            bin <auto[1],auto[0],auto[0]>                   353          1           -    Covered
            bin <auto[0],auto[0],auto[0]>                   216          1           -    Covered
    Cross Cross_rd_wr_UNDERFLOW                         100.00%        100           -    Covered
        covered/total bins:                                   5          5           -
        missing/total bins:                                   0          5           -
        % Hit:                                           100.00%        100           -
        Auto, Default and User Defined Bins:
            bin <auto[1],auto[1],auto[0]>                   418          1           -    Covered
            bin <auto[0],auto[1],auto[0]>                   485          1           -    Covered
            bin <auto[1],auto[0],auto[1]>                   110          1           -    Covered
            bin <auto[1],auto[0],auto[0]>                   302          1           -    Covered
            bin <auto[0],auto[0],auto[0]>                   286          1           -    Covered
        Illegal and Ignore Bins:
            illegal_bin high_rd_full_low_write              0                    -    ZERO
            illegal_bin high_underflow_low_read             0                    -    ZERO

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

- **Code Coverage**
  - **Branch coverage**

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

Branch Coverage for instance /FIFO_TOP/dut

    Line         Item                     Count     Source
    ----         ----                     -----     ------
  File FIFO.sv
------------------------------------IF Branch------------------------------------
    16                                     1615     Count coming in to IF
    16             1                         29             if (!FIFO_if.rst_n) begin
    21             1                        697             else if (FIFO_if.wr_en && count < FIFO_if.FIFO_DEPTH) begin
    27             1                        889             else begin
Branch totals: 3 hits of 3 branches = 100.00%

------------------------------------IF Branch------------------------------------
    30                                     889     Count coming in to IF
    30             1                        144                 if (FIFO_if.full && FIFO_if.wr_en && !FIFO_if.rd_en)
    32             1                        745                 else
Branch totals: 2 hits of 2 branches = 100.00%

------------------------------------IF Branch------------------------------------
    39                                     1615     Count coming in to IF
    39             1                         29             if (!FIFO_if.rst_n) begin
    43             1                        627             else if (FIFO_if.rd_en && count != 0) begin
    48             1                        959             else begin
Branch totals: 3 hits of 3 branches = 100.00%

------------------------------------IF Branch------------------------------------
    50                                     959     Count coming in to IF
    50             1                        110                 if (FIFO_if.empty && !FIFO_if.wr_en && FIFO_if.rd_en)
    52             1                        849                 else
Branch totals: 2 hits of 2 branches = 100.00%

------------------------------------IF Branch------------------------------------
    59                                    1339     Count coming in to IF
    59             1                         29             if (!FIFO_if.rst_n) begin
    62             1                       1310             else begin
Branch totals: 2 hits of 2 branches = 100.00%

------------------------------------IF Branch------------------------------------
    64                                    1310     Count coming in to IF
    64             1                        420                 if  ((({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b10) && !FIFO_if.full) ||
    68             1                        350                 else if ((({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b01) && !FIFO_if.empty) ||
                                            540     All False Count
Branch totals: 3 hits of 3 branches = 100.00%

------------------------------------IF Branch------------------------------------
    75                                     783     Count coming in to IF
    75             1                         77         assign FIFO_if.full = (count == FIFO_if.FIFO_DEPTH) ? 1 : 0;
    75             2                        706         assign FIFO_if.full = (count == FIFO_if.FIFO_DEPTH) ? 1 : 0;
Branch totals: 2 hits of 2 branches = 100.00%

------------------------------------IF Branch------------------------------------
    76                                     783     Count coming in to IF
    76             1                        135         assign FIFO_if.empty = (count == 0) ? 1 : 0;
    76             2                        648         assign FIFO_if.empty = (count == 0) ? 1 : 0;
Branch totals: 2 hits of 2 branches = 100.00%

------------------------------------IF Branch------------------------------------
    77                                     783     Count coming in to IF
    77             1                         92         assign FIFO_if.almostfull = (count == FIFO_if.FIFO_DEPTH-1) ? 1 : 0; // Almost full condition
    77             2                        691         assign FIFO_if.almostfull = (count == FIFO_if.FIFO_DEPTH-1) ? 1 : 0; // Almost full condition
Branch totals: 2 hits of 2 branches = 100.00%

------------------------------------IF Branch------------------------------------
    78                                     783     Count coming in to IF
    78             1                        194         assign FIFO_if.almostempty = (count == 1) ? 1 : 0; // Almost empty condition
    78             2                        589         assign FIFO_if.almostempty = (count == 1) ? 1 : 0; // Almost empty condition
Branch totals: 2 hits of 2 branches = 100.00%


Condition Coverage:
```

- **Statement Coverage**

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

Statement Coverage for instance /FIFO_TOP/dut --

    Line         Item                      Count     Source
    ----         ----                      -----     ------
  File FIFO.sv
    1                                                module FIFO(FIFO_interface.DUT FIFO_if);
    2
    3                                                    // Calculate the maximum address size for the FIFO based on its depth
    4                                                    localparam max_fifo_addr = $clog2(FIFO_if.FIFO_DEPTH);
    5
    6                                                    // Memory array to store FIFO data
    7                                                    reg [FIFO_if.FIFO_WIDTH-1:0] mem [FIFO_if.FIFO_DEPTH-1:0];
    8
    9                                                    // Write and Read pointers
    10                                                   reg [max_fifo_addr-1:0] wr_ptr, rd_ptr;
    11                                                   // Count of elements in the FIFO
    12                                                   reg [max_fifo_addr:0] count;
    13
    14                                                   // Write Operation
    15             1                          1615       always @(posedge FIFO_if.clk or negedge FIFO_if.rst_n) begin
    16                                                       if (!FIFO_if.rst_n) begin
    17             1                            29           wr_ptr <= 0;
    18             1                            29           FIFO_if.overflow <= 0; // Reset overflow condition
    19             1                            29           FIFO_if.wr_ack <= 0; // Reset write acknowledgment
    20                                                       end
    21                                                       else if (FIFO_if.wr_en && count < FIFO_if.FIFO_DEPTH) begin
    22                                                           // Write data to memory and acknowledge
    23             1                           697           mem[wr_ptr] <= FIFO_if.data_in;
    24             1                           697           FIFO_if.wr_ack <= 1;
    25             1                           697           wr_ptr <= wr_ptr + 1; // Increment write pointer
    26                                                       end
    27                                                       else begin
    28             1                           889           FIFO_if.wr_ack <= 0;
    29                                                           // Check for overflow condition when trying to write into a full FIFO
    30                                                           if (FIFO_if.full && FIFO_if.wr_en && !FIFO_if.rd_en)
    31             1                           144               FIFO_if.overflow <= 1;
    32                                                           else
    33             1                           745               FIFO_if.overflow <= 0; // Reset overflow if not full
    34                                                       end
    35                                                   end
    36
    37                                                   // Read Operation
    38             1                          1615       always @(posedge FIFO_if.clk or negedge FIFO_if.rst_n) begin
    39                                                       if (!FIFO_if.rst_n) begin
    40             1                            29           rd_ptr <= 0;
    41             1                            29           FIFO_if.underflow <= 0; // Reset underflow condition
    42                                                       end
    43                                                       else if (FIFO_if.rd_en && count != 0) begin
    44                                                           // Read data from memory
    45             1                           627           FIFO_if.data_out <= mem[rd_ptr];
    46             1                           627           rd_ptr <= rd_ptr + 1; // Increment read pointer
    47                                                       end
    48                                                       else begin
    49                                                           // Check for underflow condition when trying to read from an empty FIFO
    50                                                           if (FIFO_if.empty && !FIFO_if.wr_en && FIFO_if.rd_en)
    51             1                           110               FIFO_if.underflow <= 1;
    52                                                           else
    53             1                           849               FIFO_if.underflow <= 0; // Reset underflow if not empty
    54                                                       end
    55                                                   end
    56
    57                                                   // Count Logic
    58             1                          1339       always @(posedge FIFO_if.clk or negedge FIFO_if.rst_n) begin
    59                                                       if (!FIFO_if.rst_n) begin
    60             1                            29           count <= 0; // Initialize count on reset
    61                                                       end
    62                                                       else begin
    63                                                           // Update count based on read/write enable signals and FIFO status
    64                                                           if  ((({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b10) && !FIFO_if.full) ||
    65                                                                 (FIFO_if.empty && ({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b11)))
    66             1                           420                   count <= count + 1; // Increment count for write
    55                                                   end
    56
    57                                                   // Count Logic
    58             1                          1339       always @(posedge FIFO_if.clk or negedge FIFO_if.rst_n) begin
    59                                                       if (!FIFO_if.rst_n) begin
    60             1                            29           count <= 0; // Initialize count on reset
    61                                                       end
    62                                                       else begin
    63                                                           // Update count based on read/write enable signals and FIFO status
    64                                                           if  ((({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b10) && !FIFO_if.full) ||
    65                                                                 (FIFO_if.empty && ({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b11)))
    66             1                           420                   count <= count + 1; // Increment count for write
    67
    68                                                           else if ((({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b01) && !FIFO_if.empty) ||
    69                                                                    (FIFO_if.full && ({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b11)))
    70             1                           350                   count <= count - 1; // Decrement count for read
    71                                                       end
    72                                                   end
    73
    74                                                   // Status Signals
    75             1                           784       assign FIFO_if.full = (count == FIFO_if.FIFO_DEPTH) ? 1 : 0;
    76             1                           784       assign FIFO_if.empty = (count == 0) ? 1 : 0;
    77             1                           784       assign FIFO_if.almostfull = (count == FIFO_if.FIFO_DEPTH-1) ? 1 : 0; // Almost full condition
    78             1                           784       assign FIFO_if.almostempty = (count == 1) ? 1 : 0; // Almost empty condition

Toggle Coverage:
```

- **Toggle Coverage**

```
================================================================================
=== Instance: /FIFO_TOP/FIFO_if
=== Design Unit: work.FIFO_interface
================================================================================
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         86        86         0   100.00%

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

Toggle Coverage for instance /FIFO_TOP/FIFO_if --

                                          Node      1H->0L      0L->1H  "Coverage"
                                          ----------------------------------------
                                    almostempty           1           1      100.00
                                     almostfull           1           1      100.00
                                            clk           1           1      100.00
                                   data_in[15-0]          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
                                          rst_n           1           1      100.00
                                      underflow           1           1      100.00
                                         wr_ack           1           1      100.00
                                          wr_en           1           1      100.00

Total Node Count     =         43
Toggled Node Count   =         43
Untoggled Node Count =          0

Toggle Coverage      =     100.00% (86 of 86 bins)
```