FIFO UVM PROJECT

By: Shehab Eldeen Khaled
Supervised by: Eng. Kareem Wassem

OCTOBER 12, 2024

FIFO UVM Project

⇨ Project Overview:

This project involves the design and verification of a synchronous FIFO (First-In, First-Out) buffer, widely used in digital systems for data transfer between components operating at different clock rates or execution speeds. The design incorporates control signals like full, almost full, empty, and almost empty flags to manage data flow efficiently. Verification, conducted using Universal Verification Methodology (UVM), tested the FIFO's functionality under various conditions.

⇨ Inputs and outputs:

Port	Direction	Function
data_in	Input	Write Data: The input data bus used when writing the FIFO.
wr_en		Write Enable: If the FIFO is not full, asserting this signal causes data (on data_in) to be written into the FIFO
rd_en		Read Enable: If the FIFO is not empty, asserting this signal causes data (on data_out) to be read from the FIFO
clk		Clock signal
rst_n		Active low asynchronous reset
data_out	Output	Read Data: The sequential output data bus used when reading from the FIFO.
full		Full Flag: When asserted, this combinational output signal indicates that The FIFO is full. Write requests are ignored when the FIFO is full, initiating a write when the FIFO is full is not destructive to the contents of the FIFO.
almostfull		Almost Full: When asserted, this combinational output signal indicates that only one more write can be performed before the FIFO is full.
empty		Empty Flag: When asserted, this combinational output signal indicates that the FIFO is empty. Read requests are ignored when the FIFO is empty, initiating a read while empty is not destructive to the FIFO.
almostempty		Almost Empty: When asserted, this output combinational signal indicates that only one more read can be performed before the FIFO goes to empty.
overflow		Overflow: This sequential output signal indicates that a write request (wr_en) was rejected because the FIFO is full. Overflowing the FIFO is not destructive to the contents of the FIFO.
underflow		Underflow: This sequential output signal Indicates that the read request (rd_en) was rejected because the FIFO is empty. Under flowing the FIFO is not destructive to the FIFO.
wr_ack		Write Acknowledge: This sequential output signal indicates that a write request (wr_en) has succeeded.

⇨ ***Verification plan:***

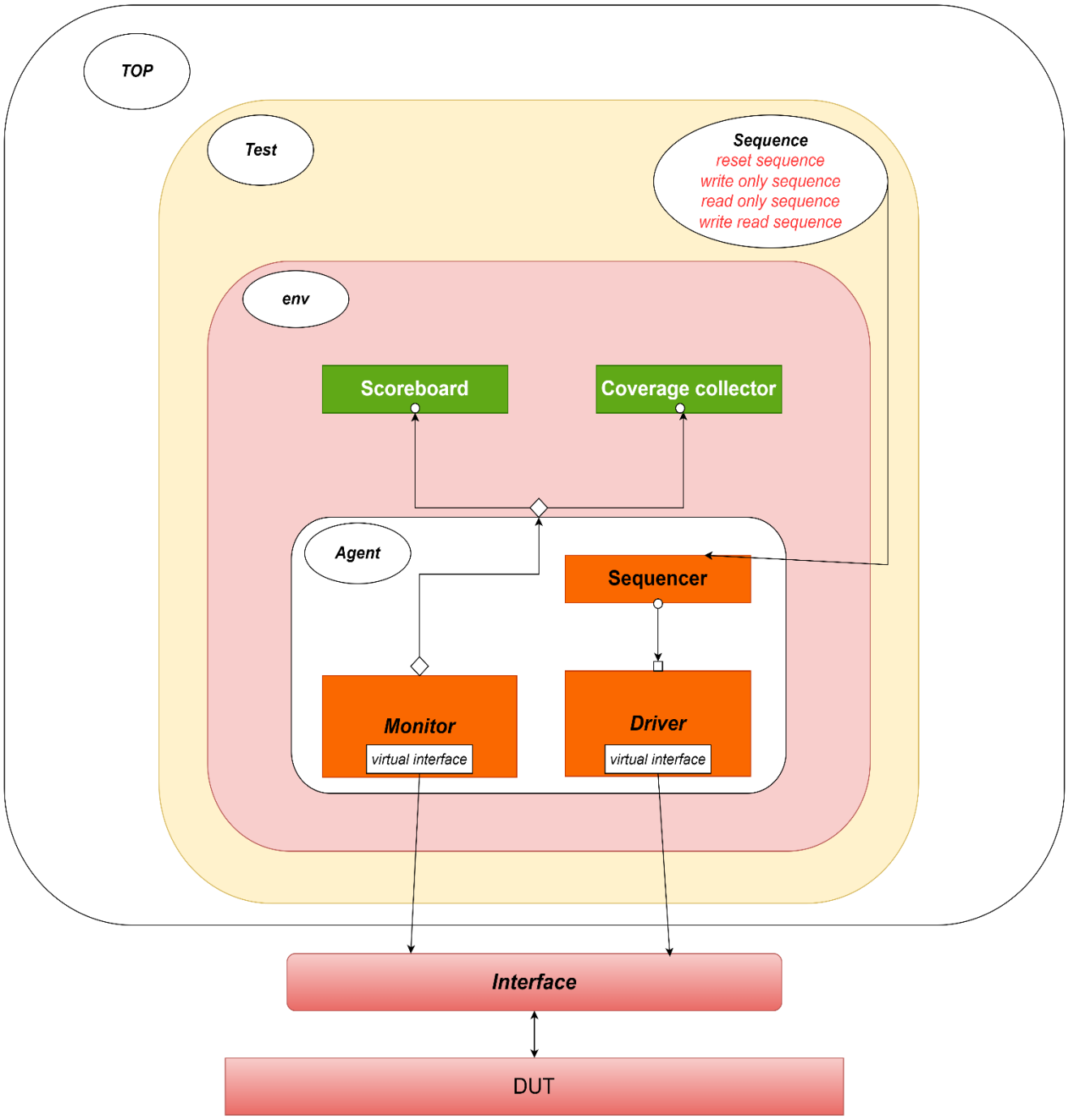
Label	Description	Stimulus Generation	Functional Coverage (Later)	Functionality Check
FIFO_1	When the reset is asserted, all outputs should be low except empty flag is high	Directed at the start of the simulation, then randomized with a constraint that make reset is off at most of simulation time	-	using immediate assertion to check the functionality of asynchronous reset, also using golden model
FIFO_2	checking when reset is disabled and write enable is high and full flag is low, write operation should be occurred and write acknowledge rises, then write pointer is incremented	Randomized with constraint that write enable is high at 50% of simulation time and low at 50% of simulation time by default and also write only constraint in the write_only sequence	cover the cases at which write enable is high and low and cover all cases between write enable, read enable, and write_ack ignoring the case at which write_en is low and write_ack is high (as write_ack is high only when write_en is high)	using concurrent assertion (wr_ack_active property and wr_ptr_ass property) to check the functionality of write operation and using golden model
FIFO_3	checking when reset is disabled and write enable is low or full flag is high, write operation shouldn't be occurred and write acknowledge will be low, then write pointer remained constant	Randomized with constraint that write enable is high at 50% of simulation time and low at 50% of simulation time by default and write only constraint in the write_only sequence	cover the cases at which write enable is high and low	using concurrent assertion (wr_ack_inactive property and wr_ptr_ass property) to check that write acknowledge is low and write pointer isn't incremented as write operation does not occur and using golden model
FIFO_4	checking overflow flag that when reset is disabled, if full flag is asserted and write enable is also asserted, then overflow flag should be high	Randomized	cover the cases by cross covering between write enable, read enable, and overflow flag ignoring the case at which write_en is low and overflow is high as it isn't important (as overflow may occur only when write_en is high)	using concurrent assertion (overflow_ass property) to check that overflow is high when both full flag and write enable are high, and using golden model
FIFO_5	checking full flag that when reset is disabled, if count reached the FIFO depth (8) , then full flag should be high and no write operation can be occurred until read operation is occurred	Randomized	cover the cases by cross covering between write enable, read enable, and full flag ignoring the case at which read_en is high and full is high as it isn't important (as full flag may occur only when write_en is high)	using immediate assertion to check that full is high when count reached FIFO depth (8), and using concurrent assertion full_inactive and full_after_almostfull and also using golden model
FIFO_6	checking almostfull flag that when reset is disabled, if count reached the FIFO depth-1 (7), then almostfull flag should be high and only one write operation can be occurred	Randomized	cover the cases by cross covering between write enable, read enable, and almostfull flag	using immediate assertion to check that almostfull is high when count reached FIFO depth-1(7), and using concurrent assertion almostfull_inactive and almostfull_from_full and using golden model

FIFO_7	checking when reset is disabled and read enable is high and empty flag is low, read operation should be occured and dataout takes the read value, then read pointer is incremented	Randomized with constraint that write enable is high at 50% of simulation time and low at 50% of simulation time by default	cover the cases at which read enable is high and low	using concurrent assertion (rd_ptr_ass property) to check the functionality of read operation and using golden model
FIFO_8	checking underflow flag that when reset is disabled, if empty flag is asserted and read enable is also asserted, then underflow flag should be high	Randomized	cover the cases by cross covering between write enable, read enable, and underflow flag ignoring the case at which read_en is low and underflow is high as it isn't important (as underflow may occur only when read_en is high)	using concurrent assertion (underflow_ass property) to check that underflow is high when both empty flag and read enable are high, and using golden model
FIFO_9	checking empty flag that when reset is disabled, if count reached zero, then empty flag should be high, and no read operation can be occurred until write operation occurred	Randomized	cover the cases by cross covering between write enable, read enable, and empty flag	using immediate assertion to check that empty is high when count reached zero, and using concurrent assertion empty_inactive and empty_from_almostempty and using golden model
FIFO_10	checking almostempty flag that when reset is disabled , if count reached zero, then almostempty flag should be high and only one read operation can be occurred	Randomized	cover the cases by cross covering between write enable , read enable , and almostempty flag	using immediate assertion to check that almostempty is high when count = 1, and using concurrent assertion almsotempty_from_empty and almsotempty_inactive and using golden model
FIFO_11	check that when reset is disabled ,the count is incremented when write opertaion is occured	Randomization	-	using concurrent assertion (count_inc_ass property) to check that count is incremented when write opertaion is occured
FIFO_12	check that when reset is disabled ,the count is decremented when read opertaion is occured	Randomization	-	using concurrent assertion (count_dec_ass property) to check that count is decremented when read opertaion is occured
FIFO_13	check that when reset is disabled ,the count remains constant when write opertaion and read operation are occured at the same time	Randomization	-	using concurrent assertion (count_const_ass property) to check that count is constant when write opertaion and read operation are occured at the same time

Feature	***Assertion***
When rst_n = 1, **count, rd_ptr** , **wr_ptr** should be zero	assert final((!FIFO_DUT.count) && (!FIFO_DUT.rd_ptr) && (!FIFO_DUT.wr_ptr))
When count reaches to FIFO_DEPTH, **full** should be high	assert final(FIFO_if.full == (FIFO_DUT.count == FIFO_DEPTH)? 1 : 0)
When count reaches FIFO_DEPTH-1, **almostfull** should be high	assert final(FIFO_if.almostfull == (FIFO_DUT.count == FIFO_DEPTH-1)? 1 : 0)
When count reaches zero, **empty** should be high	assert final(FIFO_if.empty == (FIFO_DUT.count == 0)? 1 : 0)
When count reaches 1, **almostempty** should be high	assert final(FIFO_if.almostempty == (FIFO_DUT.count == 1)? 1 : 0)
When **full** is high and read operation occurred, **full** should be low and **almostfull** should rise	@(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.full && FIFO_if.rd_en) |=> (!FIFO_if.full && $rose(FIFO_if.almostfull));
When **almostfull** is high and write operation occurred, **full** should be high and **almostfull** should fell	@(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.almostfull && FIFO_if.wr_en && !FIFO_if.rd_en) |=> ($fell(FIFO_if.almostfull) && FIFO_if.full);
When **almostfull** is high and read operation occurred only **almostfull** should be low	@(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.almostfull && FIFO_if.rd_en && !FIFO_if.wr_en) |=> !FIFO_if.almostfull;
When **full** is high and read operation occurred, **almostfull** should rise	@(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.full && FIFO_if.rd_en) |=> ($fell(FIFO_if.full) && $rose(FIFO_if.almostfull));
When **empty** is high and write operation occurred, **empty** should fell	@(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.empty && FIFO_if.wr_en) |=> ($fell(FIFO_if.empty) && $rose(FIFO_if.almostempty));
When **almostempty** is high and read operation occurred, **empty** should be high	@(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.almostempty && FIFO_if.rd_en && !FIFO_if.wr_en) |=> (FIFO_if.empty && $fell(FIFO_if.almostempty));
When **almostempty** is high and write operation occurred, **almostempty** should be low	@(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.almostempty && !FIFO_if.rd_en && FIFO_if.wr_en) |=> !FIFO_if.almostempty;
When **empty** is high and write operation occurred, **almostempty** should rise	@(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.empty && FIFO_if.wr_en && !FIFO_if.rd_en) |=> ($rose(FIFO_if.almostempty) && $fell(FIFO_if.empty));
When **full** is high and write enable is high, **overflow** should be high	@(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.full && FIFO_if.wr_en) |=> FIFO_if.overflow;

When **empty** is high and read enable is high, **underflow** should be high	@(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.empty && FIFO_if.rd_en) |=> FIFO_if.underflow;
When write operation done successfully (i.e**.** **write enable** is high and **full** is low) **wr_ack** rises	@(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (!FIFO_if.full && FIFO_if.wr_en) |=> FIFO_if.wr_ack;
When **full** is high or **write enable** is low, so **wr_ack** should be low as write operation doesn't occurred	@(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.full || !FIFO_if.wr_en) |=> !FIFO_if.wr_ack;
When write operation is done successfully, **write pointer** should be **incremented**	@(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (!FIFO_if.full && FIFO_if.wr_en) |=> FIFO_DUT.wr_ptr == $past(FIFO_DUT.wr_ptr) + 1'b1;
When read operation is done successfully, **read pointer** should be **incremented**	@(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (!FIFO_if.empty && FIFO_if.rd_en) |=> FIFO_DUT.rd_ptr == $past(FIFO_DUT.rd_ptr) + 1'b1;
When **write enable** is high, **read enable** is low, and **full** is low (i.e. write operation occurred) count should be incremented	@(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.wr_en && !FIFO_if.full && !FIFO_if.rd_en) |=> FIFO_DUT.count == $past(FIFO_DUT.count) + 1'b1;
When **read enable** is high, **write enable** is low, and **empty** is low (i.e. read operation occurred) count should be decremented	@(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (!FIFO_if.wr_en && FIFO_if.rd_en && !FIFO_if.empty) |=> FIFO_DUT.count == $past(FIFO_DUT.count) - 1'b1;
When both **read enable** and **write enable** are high, and both **full** and **empty** are low (i.e. write and read operations occurred at the same time) count should remain constant	@(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.wr_en && FIFO_if.rd_en && !FIFO_if.empty && !FIFO_if.full) |=> FIFO_DUT.count == $past(FIFO_DUT.count);

⇨ ***UVM Structure:***



⇨ ***UVM Testbench working:***

1. ***Create a top.sv file***
   - *Pass data from the interface to the DUT and the golden model.*
   - *Bind the assertion file to the design.*
   - *Set the virtual interface to the database.*
   - *Run the UVM test.*
2. **Develop a sequence_item**
   - *Randomize data inputs*
   - *make constraint blocks.*
3. ***Create three sequences to verify the design*** *(write_only , read_only , reset)*
4. ***Create a driver.sv file***
   - *Establish a virtual interface between the driver and the real interface.*
   - *Assign data from sequence Item and pass it to the interface inputs.*
5. ***Create a monitor.sv file***
   - *Establish a virtual interface between the monitor and the real interface.*
   - *Assign data from the virtual interface and pass it to the monitor object via sequence Item inputs and outputs.*
6. **Create an agent.sv**
   - *Obtain the configuration object from the database and pass it to the monitor and driver.*
   - *Establish a connection with the monitor to send data to the scoreboard and coverage collector.*
   - *Connect the sequencer to the driver.*
7. ***Create an env.sv file***
   - *Instantiate components such as the agent, scoreboard, and coverage collector.*
   - *Connect the agent to the scoreboard and coverage collector.*
8. ***Create a test.sv file***
   - *Get the virtual interface and pass it to the environment.*
   - *Instantiate sequence objects.*
   - *Instantiate the environment component.*
   - *Call the built-in function raise_objection to indicate the start of the test.*
   - *Run the sequence objects in the run_phase sequences.*
   - *Call the built-in function drop_objection to indicate the end of the test.*

⇨ ***Design before debugging:***

```
module FIFO_before_debugging(data_in, wr_en, rd_en, clk, rst_n, full, empty, almostfull,
almostempty, wr_ack, overflow, underflow, data_out);
parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;
input [FIFO_WIDTH-1:0] data_in;
input clk, rst_n, wr_en, rd_en;
output reg [FIFO_WIDTH-1:0] data_out;
output reg wr_ack, overflow;
output full, empty, almostfull, almostempty, underflow;

localparam max_fifo_addr = $clog2(FIFO_DEPTH);

reg [FIFO_WIDTH-1:0] mem [FIFO_DEPTH-1:0];

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

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

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

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

```
    if (!rst_n) begin
        count <= 0;
    end
    else begin
        if  ( ({wr_en, rd_en} == 2'b10) && !full)
            count <= count + 1;
        else if ( ({wr_en, rd_en} == 2'b01) && !empty)
            count <= count - 1;
    end
end

assign full = (count == FIFO_DEPTH)? 1 : 0;
assign empty = (count == 0)? 1 : 0;
assign underflow = (empty && rd_en)? 1 : 0;
assign almostfull = (count == FIFO_DEPTH-2)? 1 : 0;
assign almostempty = (count == 1)? 1 : 0;

endmodule
```

⇨ ***Bug report:***

1) *Almost full should be high when count = FIFO_DEPTH – 1 not count = FIFO_DEPTH – 2*
2) *Each of overflow, underflow, wr_ack, dataout signals should be zero at reset*
3) *We should give overflow signal zero when FIFO is not full and write operation is done successfully*
4) *We should give underflow signal zero when FIFO is not empty and read operation is done successfully*
5) *Underflow output is sequential output not combinational. so, it should be got from always block*
6) *At the always block of count internal signal we should take into consideration some uncovered cases:*
    a) *When wr_en and rd_en are high together and full = 1 so, count should be decremented.*
    b) *When wr_en and rd_en are high together and empty = 1 so, count should be incremented.*
    c) *When wr_en and rd_en are high together and both full and empty flags are low so, count should remain constant.*

⇨ ***Design after debugging:***

```
import shared_pkg::*;

module FIFO(FIFO_interface.DUT FIFO_if);

    localparam max_fifo_addr = $clog2(FIFO_DEPTH);

    reg [FIFO_WIDTH-1:0] mem [FIFO_DEPTH-1:0];

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

    // 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;  //fix: overflow signal should be zero at reset
            FIFO_if.wr_ack <= 0;    //fix: write_ack signal should be zero at reset
        end
        else if (FIFO_if.wr_en && count < FIFO_DEPTH) begin
            mem[wr_ptr] <= FIFO_if.data_in;
            FIFO_if.wr_ack <= 1;
            wr_ptr <= wr_ptr + 1;
            FIFO_if.overflow <= 0;  //fix: due to FIFO is not full , so overflow should be
zero
        end
        else begin
            FIFO_if.wr_ack <= 0;
            if (FIFO_if.full && FIFO_if.wr_en)
                FIFO_if.overflow <= 1;
            else
                FIFO_if.overflow <= 0;
        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;   //fix: underflow signal should be zero at reset
            FIFO_if.data_out <= 0;    //fix: dataout signal should be zero at reset
        end
        else if (FIFO_if.rd_en && count != 0) begin
            FIFO_if.data_out <= mem[rd_ptr];
            rd_ptr <= rd_ptr + 1;
            FIFO_if.underflow <= 0;    //fix: due to FIFO is not empty , so underflow
should be zero
```

```
        end

        else begin   //fix : underflow output is sequential output not combinational
            if (FIFO_if.rd_en && FIFO_if.empty) begin
                FIFO_if.underflow <= 1;       //fix
            end
            else begin
                FIFO_if.underflow <= 0;        //fix
            end
        end
    end

    always @(posedge FIFO_if.clk or negedge FIFO_if.rst_n) begin
        if (!FIFO_if.rst_n) begin
            count <= 0;
        end
        else begin
            if (({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b11) && FIFO_if.full) begin
                    count <= count-1;  //fix: when both wr_en and rd_en are high , and
full=1 , only read operation will occur
            end
            else if (({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b11) && FIFO_if.empty)
begin  //fix
                    count <= count+1; //fix: when both wr_en and rd_en are high , and
empty=1 , only write operation will occur
            end
            else if (({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b11) && !FIFO_if.full &&
!FIFO_if.empty) begin  //fix
                    count <= count; //fix: when both wr_en and rd_en are high , and both
empty=0 and full=0 , both operations (read,write) will occur
            end
            else if ( ({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b10) && !FIFO_if.full)
                count <= count + 1;
            else if ( ({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b01) && !FIFO_if.empty)
                count <= count - 1;
        end
    end

    assign FIFO_if.full = (count == FIFO_DEPTH)? 1 : 0;
    assign FIFO_if.empty = (count == 0)? 1 : 0;
    assign FIFO_if.almostfull = (count == FIFO_DEPTH-1)? 1 : 0; //fix : almostfull signal
is high when count=FIFO_DEPTH-1 not FIFO_DEPTH-2
    assign FIFO_if.almostempty = (count == 1)? 1 : 0;
endmodule
```

⇨ ***Interface:***

```
import shared_pkg::*;

interface FIFO_interface(clk);
    input clk;
    logic [FIFO_WIDTH-1:0] data_in;
    logic rst_n, wr_en, rd_en;

    logic [FIFO_WIDTH-1:0] data_out;
    logic wr_ack, overflow;
    logic wr_ack_ref, overflow_ref;

    logic [FIFO_WIDTH-1:0] data_out_ref;
    logic full, empty, almostfull, almostempty, underflow;
    logic full_ref, empty_ref, almostfull_ref, almostempty_ref, underflow_ref;

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

    modport golden (input data_in, wr_en, rd_en, clk, rst_n , output full_ref, empty_ref,
almostfull_ref, almostempty_ref, wr_ack_ref, overflow_ref, underflow_ref, data_out_ref);

endinterface //FIFO_interface
```

⇨ ***Shared package:***

```
package shared_pkg;
    parameter FIFO_WIDTH = 16;
    parameter FIFO_DEPTH = 8;
    int RD_EN_ON_DIST = 50;
    int WR_EN_ON_DIST = 50;
endpackage
```

⇨ ***Golden model:***

```
import shared_pkg::*;
module FIFO_golden (FIFO_interface.golden FIFO_if);
    reg [FIFO_WIDTH-1:0] fifo_queue [$];
    // Write operation
    always @(posedge FIFO_if.clk or negedge FIFO_if.rst_n) begin
        if (!FIFO_if.rst_n) begin
            fifo_queue.delete();
            FIFO_if.wr_ack_ref <= 0;
            FIFO_if.overflow_ref <= 0;
        end
        else if (FIFO_if.wr_en && !FIFO_if.full_ref) begin
            fifo_queue.push_back(FIFO_if.data_in);
            FIFO_if.wr_ack_ref <= 1;
            FIFO_if.overflow_ref <= 0;
        end
        else begin
            FIFO_if.wr_ack_ref <= 0;
            if (FIFO_if.full_ref && FIFO_if.wr_en)
                FIFO_if.overflow_ref <= 1;
            else
                FIFO_if.overflow_ref <= 0;
        end
    end
    // Read operation
    always @(posedge FIFO_if.clk or negedge FIFO_if.rst_n) begin
        if (!FIFO_if.rst_n) begin
            fifo_queue.delete();
            FIFO_if.underflow_ref <= 0;
            FIFO_if.data_out_ref <= 0;
        end
        else if (FIFO_if.rd_en && !FIFO_if.empty_ref) begin
            FIFO_if.data_out_ref <= fifo_queue.pop_front();
            FIFO_if.underflow_ref <= 0;
        end
        else begin
            if (FIFO_if.empty_ref && FIFO_if.rd_en)
                FIFO_if.underflow_ref <= 1;
            else
                FIFO_if.underflow_ref <= 0;
        end
    end
    assign FIFO_if.full_ref = (fifo_queue.size() >= FIFO_DEPTH )? 1:0;
    assign FIFO_if.almostfull_ref = (fifo_queue.size() == FIFO_DEPTH-1)? 1:0;
    assign FIFO_if.empty_ref = (fifo_queue.size() == 0)? 1:0;
    assign FIFO_if.almostempty_ref = (fifo_queue.size() == 1)? 1:0;
endmodule
```

⇨ ***Assertions file:***

```
import shared_pkg::*;

module FIFO_SVA (FIFO_interface.DUT FIFO_if);

    `ifdef FIFO_Assertions

    //immediate assertions (combinational outputs)
    always_comb begin
        if(!FIFO_if.rst_n) begin
            reset_ass: assert final((!FIFO_DUT.count) && (!FIFO_DUT.rd_ptr) &&
(!FIFO_DUT.wr_ptr))
            else $display("at time: %t , reset fails",$time);
        end
        full_ass:         assert final(FIFO_if.full == (FIFO_DUT.count == FIFO_DEPTH)? 1 :
0)         else $display("at time: %t , full fails",$time);
        empty_ass:        assert final(FIFO_if.empty == (FIFO_DUT.count == 0)? 1 :
0)                 else $display("at time: %t , empty fails",$time);
        almostfull_ass:   assert final(FIFO_if.almostfull == (FIFO_DUT.count ==
FIFO_DEPTH-1)? 1 : 0) else $display("at time: %t , almost full fails",$time);
        almost_empty_ass: assert final(FIFO_if.almostempty == (FIFO_DUT.count == 1)? 1 : 0
)          else $display("at time: %t , almost empty fails",$time);
    end

    //concurrent assertions
    // full signal
    property full_inactive;
        @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.full && FIFO_if.rd_en)
|=> (!FIFO_if.full && $rose(FIFO_if.almostfull));
    endproperty

    property full_after_almostfull;
        @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.almostfull &&
FIFO_if.wr_en && !FIFO_if.rd_en) |=> ($fell(FIFO_if.almostfull) && FIFO_if.full);
    endproperty

    // almost full signal
    property almostfull_from_full;
        @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.full && FIFO_if.rd_en)
|=> ($fell(FIFO_if.full) && $rose(FIFO_if.almostfull));
    endproperty

    property almostfull_inactive;
        @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.almostfull &&
FIFO_if.rd_en && !FIFO_if.wr_en) |=> !FIFO_if.almostfull;
    endproperty
```

```
    // empty signal
    property empty_inactive;
        @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.empty &&
FIFO_if.wr_en) |=> ($fell(FIFO_if.empty) && $rose(FIFO_if.almostempty));
    endproperty

    property empty_from_almostempty;
        @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.almostempty &&
FIFO_if.rd_en && !FIFO_if.wr_en) |=> (FIFO_if.empty && $fell(FIFO_if.almostempty));
    endproperty

    // almost empty signal
    property almsotempty_inactive;
        @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.almostempty &&
!FIFO_if.rd_en && FIFO_if.wr_en) |=> !FIFO_if.almostempty;
    endproperty

    property almsotempty_from_empty;
        @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.empty && FIFO_if.wr_en
&& !FIFO_if.rd_en) |=> ($rose(FIFO_if.almostempty) && $fell(FIFO_if.empty));
    endproperty


    // overflow signal
    property overflow_ass;
        @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.full && FIFO_if.wr_en)
|=> FIFO_if.overflow;
    endproperty

    // underflow signal
    property underflow_ass;
        @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.empty &&
FIFO_if.rd_en) |=> FIFO_if.underflow;
    endproperty

    // wr_ack signal
    property wr_ack_active;
        @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (!FIFO_if.full &&
FIFO_if.wr_en) |=> FIFO_if.wr_ack;
    endproperty

    property wr_ack_inactive;
        @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.full ||
!FIFO_if.wr_en) |=> !FIFO_if.wr_ack;
    endproperty

    // write pointer internal signal
    property wr_ptr_ass;
```

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

    // read pointer internal signal
    property rd_ptr_ass;
        @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (!FIFO_if.empty &&
FIFO_if.rd_en) |=> FIFO_DUT.rd_ptr == $past(FIFO_DUT.rd_ptr) + 1'b1;
    endproperty

    // counter internal signal
    property count_inc_ass;
        @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.wr_en && !FIFO_if.full
&& !FIFO_if.rd_en) |=> FIFO_DUT.count == $past(FIFO_DUT.count) + 1'b1;
    endproperty

    property count_dec_ass;
        @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (!FIFO_if.wr_en &&
FIFO_if.rd_en && !FIFO_if.empty) |=> FIFO_DUT.count == $past(FIFO_DUT.count) - 1'b1;
    endproperty

    property count_const_ass;
        @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.wr_en && FIFO_if.rd_en
&& !FIFO_if.empty && !FIFO_if.full) |=> FIFO_DUT.count == $past(FIFO_DUT.count);
    endproperty


    // assert properties
    full_inactive_assert: assert property(full_inactive) else $display("at time %t : full
inactive Fails",$time);
    full_after_almostfull_assert: assert property(full_after_almostfull) else $display("at
time %t : full_after_almostfull Fails",$time);
    almostfull_from_full_assert: assert property(almostfull_from_full) else $display("at
time %t : almostfull_from_full Fails",$time);
    almostfull_inactive_assert: assert property(almostfull_inactive) else $display("at
time %t : almostfull_inactive Fails",$time);
    empty_inactive_assert: assert property(empty_inactive) else $display("at time %t :
empty_inactive Fails",$time);
    empty_from_almostempty_assert: assert property(empty_from_almostempty) else
$display("at time %t : empty_from_almostempty Fails",$time);
    almsotempty_inactive_assert: assert property(almsotempty_inactive) else $display("at
time %t : almsotempty_inactive Fails",$time);
    almsotempty_from_empty_assert: assert property(almsotempty_from_empty) else
$display("at time %t : almsotempty_from_empty Fails",$time);
    overflow_assert: assert property(overflow_ass) else $display("at time %t : overflow
Fails",$time);
    underflow_assert: assert property(underflow_ass) else $display("at time %t : underflow
Fails",$time);
```

```
    wr_ack_assert: assert property(wr_ack_active) else $display("at time %t : write ack
Fails",$time);
    wr_ack_inactive_assert: assert property(wr_ack_inactive) else $display("at time %t :
write ack Fails",$time);
    wr_ptr_assert: assert property(wr_ptr_ass) else $display("at time %t : write pointer
Fails",$time);
    rd_ptr_assert: assert property(rd_ptr_ass) else $display("at time %t : read pointer
Fails",$time);
    count_inc_assert: assert property(count_inc_ass) else $display("at time %t : counter
increment Fails",$time);
    count_dec_assert: assert property(count_dec_ass) else $display("at time %t : counter
decrement Fails",$time);
    count_const_assert: assert property(count_const_ass) else $display("at time %t :
counter const Fails",$time);

    // cover properties
    full_inactive_cover: cover property(full_inactive);
    full_after_almostfull_cover: cover property(full_after_almostfull);
    almostfull_from_full_cover: cover property(almostfull_from_full);
    almostfull_inactive_cover: cover property(almostfull_inactive);
    empty_inactive_cover: cover property(empty_inactive);
    empty_from_almostempty_cover: cover property(empty_from_almostempty);
    almsotempty_inactive_cover: cover property(almsotempty_inactive);
    almsotempty_from_empty_cover: cover property(almsotempty_from_empty);
    overflow_cover: cover property (overflow_ass);
    underflow_cover: cover property (underflow_ass);
    wr_ack_cover: cover property(wr_ack_active);
    wr_ack_inactive_cover: cover property(wr_ack_inactive);
    wr_ptr_cover: cover property (wr_ptr_ass);
    rd_ptr_cover: cover property (rd_ptr_ass);
    count_inc_cover: cover property (count_inc_ass);
    count_dec_cover: cover property (count_dec_ass);
    count_const_cover: cover property (count_const_ass);
    `endif
endmodule
```

⇨ ***UVM Packages:***

1) ***Configuration package:***

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

    class FIFO_config extends uvm_object;
        `uvm_object_utils(FIFO_config)
        virtual FIFO_interface FIFO_vif;

        // constructor
        function new(string name = "FIFO_config");
            super.new(name);
        endfunction
    endclass
endpackage
```

2) ***Sequencer package:***

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

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

        // constructor
        function new(string name = "FIFO_sequencer" , uvm_component parent = null);
            super.new(name,parent);
        endfunction
    endclass
endpackage
```

3) ***Sequence item package:***

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

    class FIFO_seq_item extends uvm_sequence_item;
        `uvm_object_utils(FIFO_seq_item)

        // Define signals of sequence items
        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 wr_ack_ref, overflow_ref;

        logic [FIFO_WIDTH-1:0] data_out_ref;
        logic full, empty, almostfull, almostempty, underflow;
        logic full_ref, empty_ref, almostfull_ref, almostempty_ref, underflow_ref;

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

        // print all inputs and outputs
        function string convert2string();
            return $sformatf("%s rst_n= %b , wr_en= %b , rd_en= %b , data_in= %0h ,
data_out = %0h, wr_ack = %0b, full = %0b, empty = %0b, underflow = %0b, almostempty = %0b,
almostfull = %0b, overflow = %0b"
            ,super.convert2string(),rst_n,wr_en,rd_en,data_in,data_out,wr_ack,full,empty,u
nderflow,almostempty,almostfull,overflow);
        endfunction

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

        // print reference outputs
        function string convert2string_ref();
            return $sformatf("data_out_ref= %0h, wr_ack_ref = %0b, full_ref = %0b,
empty_ref = %0b, underflow_ref = %0b, almostempty_ref = %0b, almostfull_ref = %0b,
overflow_ref = %0b"
```

```
            ,data_out_ref,wr_ack_ref,full_ref,empty_ref,underflow_ref,almostempty_ref,almo
stfull_ref,overflow_ref);
        endfunction

        /////////////// constraints ///////////////////
        ////////////////////////////////////////////////
        constraint reset_c {
            rst_n dist {1:=96 , 0:=4};
        }

        // write and read with different probabilities ==> for write_read sequence
        constraint write_and_read {
            wr_en dist {1:= WR_EN_ON_DIST , 0:= 100-WR_EN_ON_DIST};
            rd_en dist {1:= RD_EN_ON_DIST , 0:= 100-RD_EN_ON_DIST};
        }

        // only read constraint for read_only sequence
        constraint read_only {
            rd_en == 1;
            wr_en == 0;
        }

         // only write constraint for write_only sequence
        constraint write_only {
            rd_en == 0;
            wr_en == 1;
        }
    endclass
endpackage
```

4) ***Reset sequence package:***

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

    class FIFO_rst_sequence extends uvm_sequence #(FIFO_seq_item);
        `uvm_object_utils(FIFO_rst_sequence)
        FIFO_seq_item rst_seq;

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

        task body ();
            rst_seq = FIFO_seq_item::type_id::create("rst_seq"); // create a sequence item
            start_item(rst_seq);
            rst_seq.rst_n = 0;
            rst_seq.wr_en = 1;
            rst_seq.rd_en = 0;
            rst_seq.data_in = 5;
            finish_item(rst_seq);
        endtask
    endclass

endpackage
```

5) ***Write only sequence package:***

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

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

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

        task body();
            repeat(1000) begin
                write_only_seq = FIFO_seq_item::type_id::create("write_only_seq"); //
create a sequence item

                /////////////// Edit Constraints /////////////////
                ////////////////////////////////////////////////
                write_only_seq.constraint_mode(0);  // disable all constraints
                write_only_seq.write_only.constraint_mode(1);  // enable write only
constraint
                write_only_seq.reset_c.constraint_mode(1);   // enable reset constraint
                start_item(write_only_seq);
                assert(write_only_seq.randomize());
                finish_item(write_only_seq);
            end
        endtask
    endclass

endpackage
```

6) ***Read only sequence package:***

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

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

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

        task body();
            repeat(1000) begin
                read_only_seq = FIFO_seq_item::type_id::create("read_only_seq"); // create
a sequence item

                /////////////// Edit Constraints /////////////////
                ////////////////////////////////////////////////
                read_only_seq.constraint_mode(0);  // disable all constraints
                read_only_seq.read_only.constraint_mode(1);  // enable read only
constraint
                read_only_seq.reset_c.constraint_mode(1);   // enable reset constraint
                start_item(read_only_seq);
                assert(read_only_seq.randomize());
                finish_item(read_only_seq);
            end
        endtask
    endclass

endpackage
```

7) ***Read and write sequence package:***

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

    class FIFO_read_write_sequence extends uvm_sequence #(FIFO_seq_item);
        `uvm_object_utils(FIFO_read_write_sequence)
        FIFO_seq_item read_write_seq;

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

        task body();
            repeat(1000) begin
                read_write_seq = FIFO_seq_item::type_id::create("read_write_seq"); //
create a sequence item

                /////////////// Edit Constraints /////////////////
                ////////////////////////////////////////////////
                read_write_seq.constraint_mode(0);  // disable all constraints
                read_write_seq.write_and_read.constraint_mode(1);  // enable read_write
constraint
                read_write_seq.reset_c.constraint_mode(1);   // enable reset constraint
                start_item(read_write_seq);
                assert(read_write_seq.randomize());
                finish_item(read_write_seq);
            end
        endtask
    endclass

endpackage
```

8) ***Driver package:***

```
package FIFO_driver_pkg;
    import shared_pkg::*;
    import uvm_pkg::*;
    import FIFO_seq_item_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 stim_seq_item;

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

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

            // generate stimulus
            forever begin
                stim_seq_item = FIFO_seq_item::type_id::create("stim_seq_item");
                seq_item_port.get_next_item(stim_seq_item);

                // generate inputs
                FIFO_vif.rst_n = stim_seq_item.rst_n;
                FIFO_vif.wr_en = stim_seq_item.wr_en;
                FIFO_vif.rd_en = stim_seq_item.rd_en;
                FIFO_vif.data_in = stim_seq_item.data_in;
                @(negedge FIFO_vif.clk);
                seq_item_port.item_done();
                `uvm_info("run_phase", stim_seq_item.convert2string_stimulus(), UVM_HIGH)
            end
        endtask
    endclass
endpackage
```

9) ***Monitor package:***

```
package FIFO_monitor_pkg;
    import shared_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 mon_seq_item;
        uvm_analysis_port #(FIFO_seq_item) mon_ap;
        function new(string name = "FIFO_monitor" , uvm_component parent = null);
            super.new(name,parent);
        endfunction
        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            mon_ap = new("mon_ap",this);
        endfunction
        task run_phase (uvm_phase phase);
            super.run_phase(phase);
            forever begin
                mon_seq_item = FIFO_seq_item::type_id::create("mon_seq_item");
                @(negedge FIFO_vif.clk);
                mon_seq_item.rst_n = FIFO_vif.rst_n;
                mon_seq_item.wr_en = FIFO_vif.wr_en;
                mon_seq_item.rd_en = FIFO_vif.rd_en;
                mon_seq_item.data_in = FIFO_vif.data_in;
                mon_seq_item.data_out = FIFO_vif.data_out;
                mon_seq_item.wr_ack = FIFO_vif.wr_ack;
                mon_seq_item.overflow = FIFO_vif.overflow;
                mon_seq_item.underflow = FIFO_vif.underflow;
                mon_seq_item.full = FIFO_vif.full;
                mon_seq_item.empty = FIFO_vif.empty;
                mon_seq_item.almostfull = FIFO_vif.almostfull;
                mon_seq_item.almostempty = FIFO_vif.almostempty;
                mon_seq_item.data_out_ref = FIFO_vif.data_out_ref;
                mon_seq_item.wr_ack_ref = FIFO_vif.wr_ack_ref;
                mon_seq_item.overflow_ref = FIFO_vif.overflow_ref;
                mon_seq_item.underflow_ref = FIFO_vif.underflow_ref;
                mon_seq_item.full_ref = FIFO_vif.full_ref;
                mon_seq_item.empty_ref = FIFO_vif.empty_ref;
                mon_seq_item.almostfull_ref = FIFO_vif.almostfull_ref;
                mon_seq_item.almostempty_ref = FIFO_vif.almostempty_ref;
                mon_ap.write(mon_seq_item);
                `uvm_info("run_phase", mon_seq_item.convert2string(), UVM_HIGH)
            end
        endtask
```

```
    endclass
endpackage
```

10) ***Monitor package:***

```
package FIFO_agent_pkg;
    import shared_pkg::*;
    import uvm_pkg::*;
    import FIFO_seq_item_pkg::*;
    import FIFO_config_pkg::*;
    import FIFO_monitor_pkg::*;
    import FIFO_driver_pkg::*;
    import FIFO_sequencer_pkg::*;
    `include "uvm_macros.svh"
    class FIFO_agent extends uvm_agent;
        `uvm_component_utils(FIFO_agent)
        FIFO_monitor mon;
        FIFO_sequencer sqr;
        FIFO_driver drv;
        FIFO_config FIFO_cfg;
        uvm_analysis_port #(FIFO_seq_item) agt_ap;
        function new(string name = "FIFO_agent" , uvm_component parent = null);
            super.new(name,parent);
        endfunction
        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            if (!uvm_config_db #(FIFO_config)::get(this , "" , "CFG" , FIFO_cfg)) begin
                `uvm_fatal("build_phase", "Driver - Unable to get configuration object");
            end
            agt_ap = new("agt_ap",this);
            sqr = FIFO_sequencer::type_id::create("sqr",this);
            mon = FIFO_monitor::type_id::create("mon",this);
            drv = FIFO_driver::type_id::create("drv",this);
        endfunction
        function void connect_phase(uvm_phase phase);
            drv.FIFO_vif = FIFO_cfg.FIFO_vif;
            mon.FIFO_vif = FIFO_cfg.FIFO_vif;
            drv.seq_item_port.connect(sqr.seq_item_export);
            mon.mon_ap.connect(agt_ap);
        endfunction
    endclass
endpackage
```

11) Coverage package:

```
package FIFO_coverage_pkg;
    import shared_pkg::*;
    import uvm_pkg::*;
    import FIFO_seq_item_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 cov_seq_item;
        /////////////// covergroups ///////////////////////
        covergroup write_read_cover;
            rst_cvg:          coverpoint cov_seq_item.rst_n; //coverpoint for rst_n signal
            write_enable_cvg: coverpoint cov_seq_item.wr_en; //coverpoint for write_en
signal
            read_enable_cvg : coverpoint cov_seq_item.rd_en;//coverpoint for read_en
signal
            full_cvg:         coverpoint cov_seq_item.full;//coverpoint for full flag
output
            empty_cvg:        coverpoint cov_seq_item.empty;//coverpoint for empty flag
output
            almost_full_cvg:  coverpoint cov_seq_item.almostfull;     //coverpoint for
almostfull flag output
            almost_empty_cvg: coverpoint cov_seq_item.almostempty;    //coverpoint for
almostempty flag output
            write_ack_cvg:    coverpoint cov_seq_item.wr_ack;         //coverpoint for
write_ack flag output
            overflow_cvg:     coverpoint cov_seq_item.overflow;       //coverpoint for
overflow flag output
            underflow_cvg:    coverpoint cov_seq_item.underflow;      //coverpoint for
underflow flag output
            write_read_full:        cross
write_enable_cvg,read_enable_cvg,full_cvg{                     // cross between wr_en ,
rd_en , full
                //not important for full output if read_en = 1 (as full=1 may only when
wr_en=1)
                ignore_bins full_read_en00 = binsof(read_enable_cvg) intersect {1} &&
binsof(full_cvg) intersect {1};
            }
            write_read_empty:       cross write_enable_cvg,read_enable_cvg,empty_cvg;//
cross between wr_en , rd_en ,empty
            write_read_almost_full: cross
write_enable_cvg,read_enable_cvg,almost_full_cvg;// cross between wr_en , rd_en ,
almostfull
```

```
            write_read_almostempty: cross
write_enable_cvg,read_enable_cvg,almost_empty_cvg;        // cross between wr_en , rd_en ,
almostempty
            write_read_wr_ack:      cross
write_enable_cvg,read_enable_cvg,write_ack_cvg{            // cross between wr_en , rd_en
, wr_ack
                //not important for wr_ack output if write_en = 0 (as wr_ack=1 only when
wr_en=1)
                ignore_bins wr_ack_wr_en00 = binsof(write_enable_cvg) intersect {0} &&
binsof(write_ack_cvg) intersect {1};
            }
            write_read_overflow:    cross
write_enable_cvg,read_enable_cvg,overflow_cvg{             // cross between wr_en , rd_en
, overflow
                //not important for overflow output if write_en = 0 (as overflow occurs
only when wr_en=1)
                ignore_bins write_overflow00 = binsof(write_enable_cvg) intersect {0} &&
binsof(overflow_cvg) intersect {1};
            }
            write_read_underflow:   cross
write_enable_cvg,read_enable_cvg,underflow_cvg{            // cross between wr_en , rd_en
, underflow
                //not important for underflow output if read_en = 0 (as underflow occurs
only when rd_en=1)
                ignore_bins read_underflow00 = binsof(read_enable_cvg) intersect {0} &&
binsof(underflow_cvg) intersect {1};
            }
        endgroup
        function new(string name = "FIFO_coverage" , uvm_component parent = null);
            super.new(name,parent);
            write_read_cover = new();
        endfunction
        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            cov_export = new("cov_export",this);
            cov_fifo = new("cov_fifo",this);
        endfunction
        function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
            cov_export.connect(cov_fifo.analysis_export);
        endfunction
        task run_phase(uvm_phase phase);
            super.run_phase(phase);
            forever begin
                cov_fifo.get(cov_seq_item);
                write_read_cover.sample();
            end
        endtask
```

```
    endclass
endpackage
```

12) ***Scoreboard package:***

```
package FIFO_scoreboard_pkg;
    import shared_pkg::*;
    import uvm_pkg::*;
    import FIFO_seq_item_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 sb_seq_item;
    int correct_count = 0;
    int error_count = 0;

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

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

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

    //run_phase
    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever begin
            sb_fifo.get(sb_seq_item);

            //checking
            if (sb_seq_item.data_out != sb_seq_item.data_out_ref
            || sb_seq_item.wr_ack != sb_seq_item.wr_ack_ref
```

```
            || sb_seq_item.overflow != sb_seq_item.overflow_ref
            || sb_seq_item.underflow != sb_seq_item.underflow_ref
            || sb_seq_item.full != sb_seq_item.full_ref
            || sb_seq_item.empty != sb_seq_item.empty_ref
            || sb_seq_item.almostfull != sb_seq_item.almostfull_ref
            || sb_seq_item.almostempty != sb_seq_item.almostempty_ref) begin

                `uvm_error("run_phase" , $sformatf("comparison failed , transaction
recieved by the DUT: %0s\n , while the correct outputs: %0s "
                ,sb_seq_item.convert2string() , sb_seq_item.convert2string_ref()));
                error_count++;
            end
            else begin
                correct_count++;
            end
        end
    endtask

    // report_phase
    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
```

13) ***Environment package:***

```
package FIFO_env_pkg;
    import shared_pkg::*;
    import uvm_pkg::*;
    import FIFO_agent_pkg::*;
    import FIFO_scoreboard_pkg::*;
    import FIFO_coverage_pkg::*;
    `include "uvm_macros.svh"
    class FIFO_env extends uvm_env;
        `uvm_component_utils(FIFO_env)
        FIFO_agent agt;
        FIFO_scoreboard sb;
        FIFO_coverage cov;

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

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

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

14) ***Test package:***

```
package FIFO_test_pkg;
    import uvm_pkg::*;
    import shared_pkg::*;
    import FIFO_env_pkg::*;
    import FIFO_read_only_sequence_pkg::*;
    import FIFO_write_only_sequence_pkg::*;
    import FIFO_read_write_sequence_pkg::*;
    import FIFO_rst_sequence_pkg::*;
    import FIFO_config_pkg::*;
    `include "uvm_macros.svh"

    class FIFO_test extends uvm_test;
        `uvm_component_utils(FIFO_test)
        FIFO_env env;
        virtual FIFO_interface FIFO_vif;
        FIFO_config FIFO_cfg;
        FIFO_rst_sequence rst_seq;
        FIFO_read_only_sequence rd_seq;
        FIFO_write_only_sequence wr_seq;
        FIFO_read_write_sequence rd_wr_seq;

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

        //build phase
        function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            env = FIFO_env::type_id::create("env",this);
            FIFO_cfg = FIFO_config::type_id::create("FIFO_cfg",this);
            rst_seq = FIFO_rst_sequence::type_id::create("rst_seq",this);
            rd_seq = FIFO_read_only_sequence::type_id::create("rd_seq",this);
            wr_seq = FIFO_write_only_sequence::type_id::create("wr_seq",this);
            rd_wr_seq = FIFO_read_write_sequence::type_id::create("rd_wr_seq",this);

            if (!uvm_config_db #(virtual FIFO_interface)::get(this , "" , "FIFO_IF" ,
FIFO_cfg.FIFO_vif)) begin
                `uvm_fatal("build_phase" , "Test - unable to get the virtual interface of
FIFO from uvm_config_db");
            end
            uvm_config_db #(FIFO_config)::set(this , "*" , "CFG" , FIFO_cfg);
        endfunction

        //run phase
        task run_phase (uvm_phase phase);
```

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

            //reset seq
            `uvm_info("run_phase", "reset_asserted" , UVM_LOW)
            rst_seq.start(env.agt.sqr);
            `uvm_info("run_phase" , "reset_deasserted" , UVM_LOW)

            //write_only sequence
            `uvm_info("run_phase", "stimulus generation started" , UVM_LOW)
            wr_seq.start(env.agt.sqr);
            `uvm_info("run_phase", "stimulus generation ended" , UVM_LOW)

            //read_only sequence
            `uvm_info("run_phase", "stimulus generation started" , UVM_LOW)
            rd_seq.start(env.agt.sqr);
            `uvm_info("run_phase", "stimulus generation ended" , UVM_LOW)

            // write_read sequence
            `uvm_info("run_phase", "stimulus generation started" , UVM_LOW)
            rd_wr_seq.start(env.agt.sqr);
            `uvm_info("run_phase", "stimulus generation ended" , UVM_LOW)

            phase.drop_objection(this);
        endtask

    endclass

endpackage
```

⇨ ***Top file:***

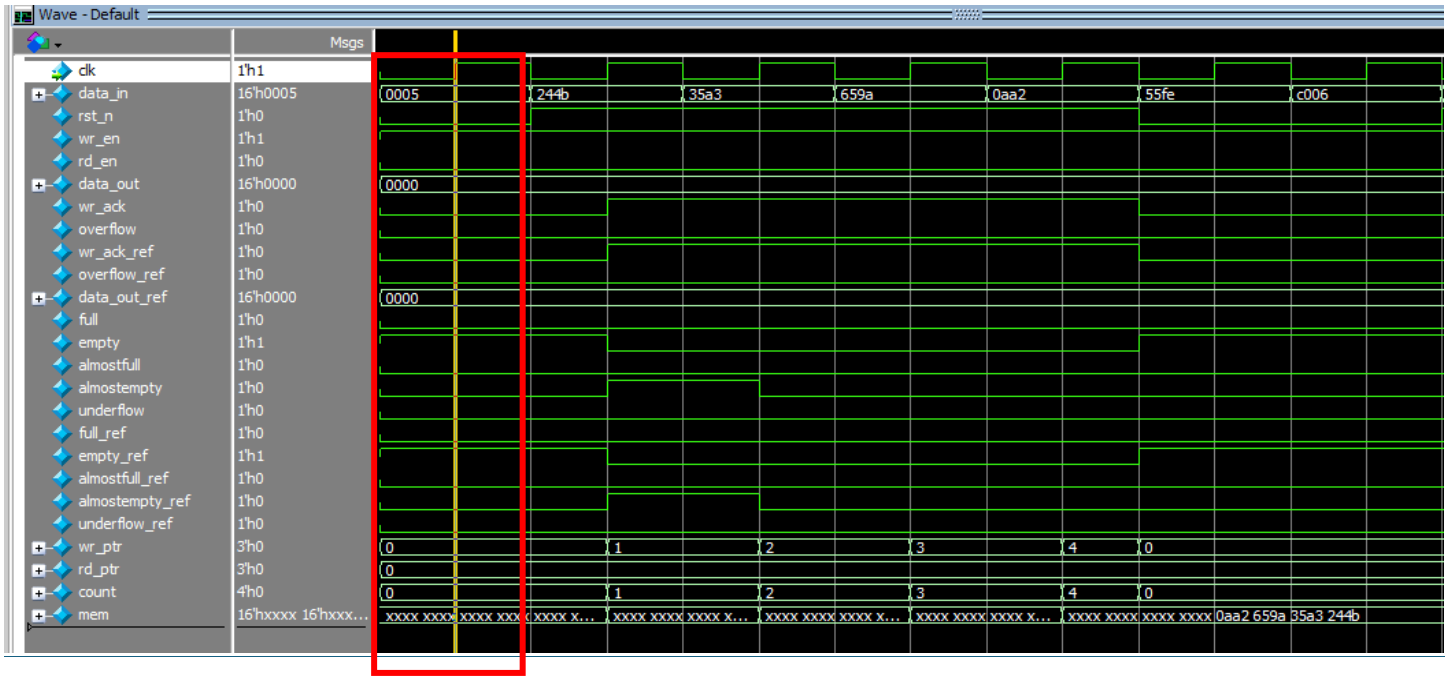
```
import uvm_pkg::*;
import FIFO_test_pkg::*;
`include "uvm_macros.svh"

module FIFO_top ();
    logic clk;
    initial begin
        clk = 0;
        forever begin
            #1 clk = ~clk;
        end
    end

    FIFO_interface FIFO_if(clk);
    FIFO FIFO_DUT(FIFO_if);
    FIFO_golden ref_model(FIFO_if);
    bind FIFO FIFO_SVA FIFO_SVA_INST(FIFO_if);

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

endmodule
```

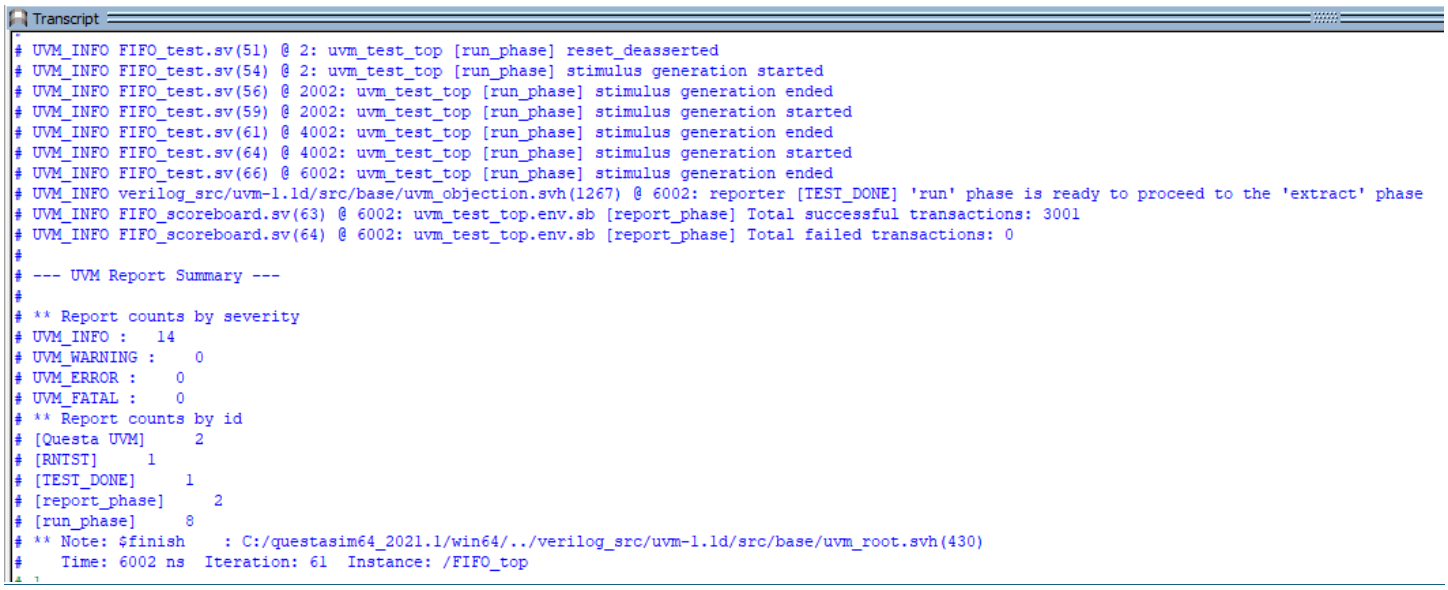
⇨ ***Do file:***

```
vlib work
vlog -f src_files.list +define+FIFO_Assertions +cover -covercells
vsim -voptargs=+acc work.FIFO_top -cover -classdebug -uvmcontrol=all
add wave /FIFO_top/FIFO_if/*
coverage save FIFO_tb.ucdb -onexit
run -all
```

⇨ ***Questasim snippets:***

1) Reset sequence:

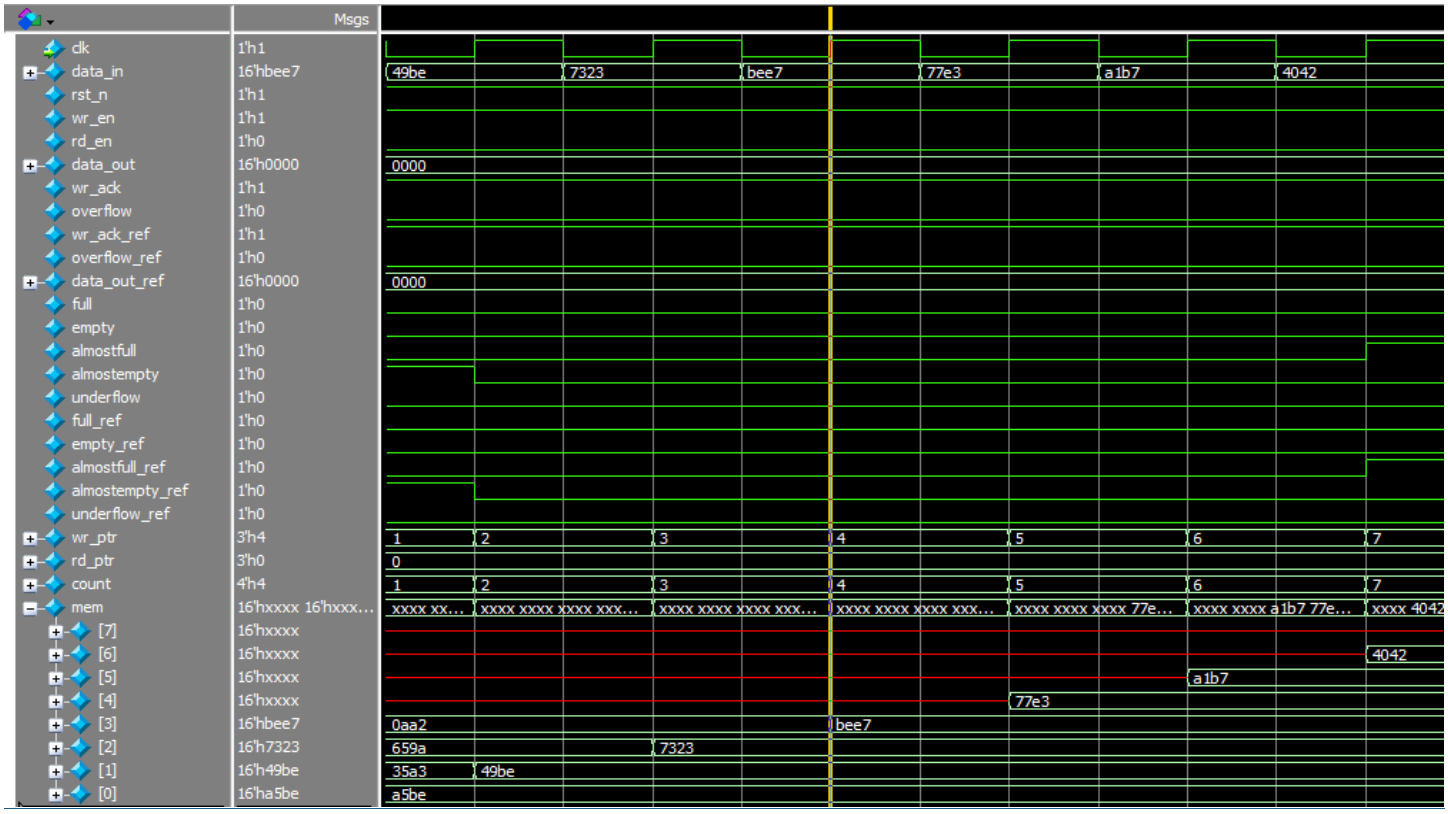




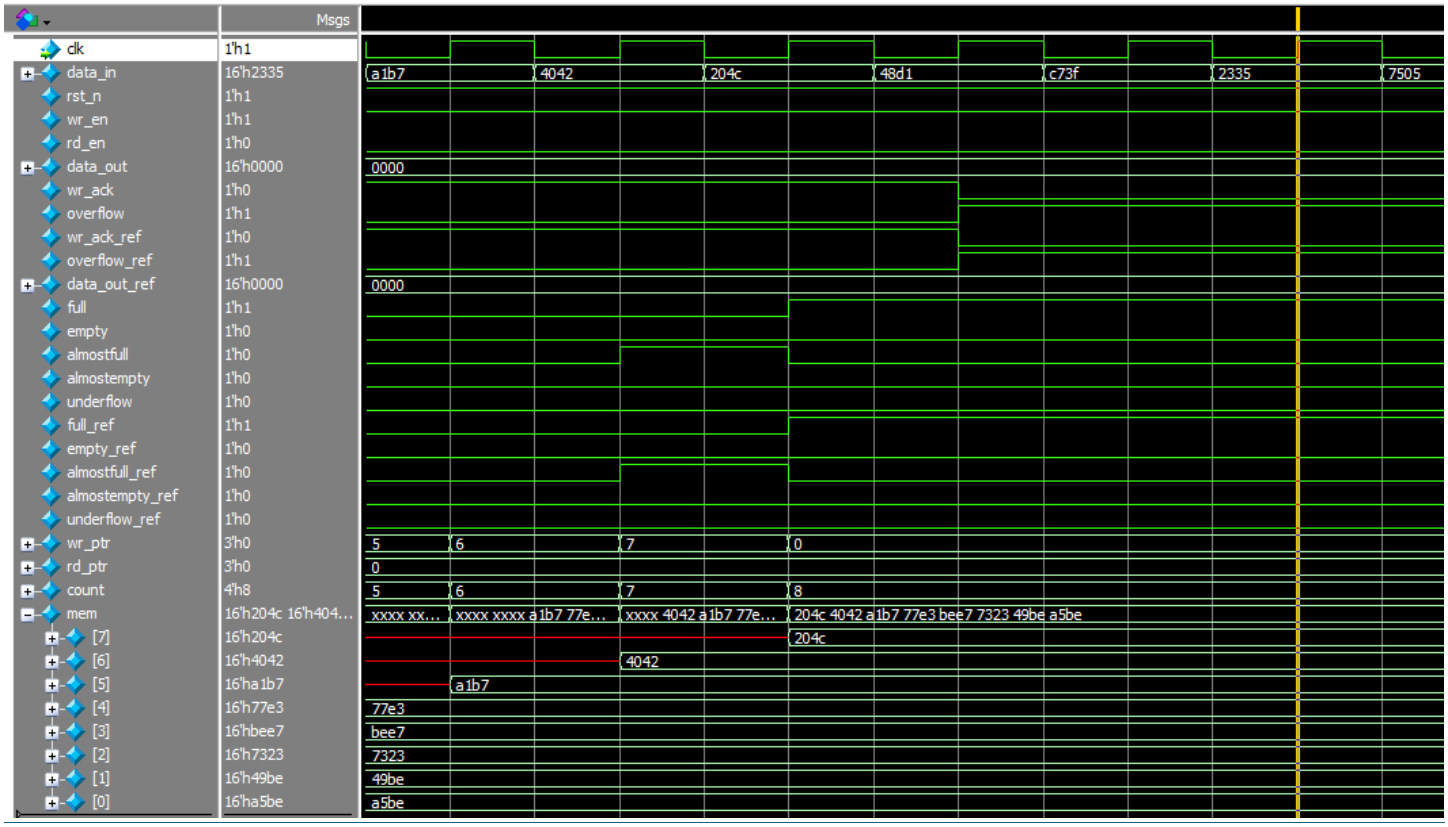
```
Transcript
# UVM_INFO FIFO_test.sv(51) @ 2: uvm_test_top [run_phase] reset_deasserted
# UVM_INFO FIFO_test.sv(54) @ 2: uvm_test_top [run_phase] stimulus generation started
# UVM_INFO FIFO_test.sv(56) @ 2002: uvm_test_top [run_phase] stimulus generation ended
# UVM_INFO FIFO_test.sv(59) @ 2002: uvm_test_top [run_phase] stimulus generation started
# UVM_INFO FIFO_test.sv(61) @ 4002: uvm_test_top [run_phase] stimulus generation ended
# UVM_INFO FIFO_test.sv(64) @ 4002: uvm_test_top [run_phase] stimulus generation started
# UVM_INFO FIFO_test.sv(66) @ 6002: uvm_test_top [run_phase] stimulus generation ended
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 6002: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO FIFO_scoreboard.sv(63) @ 6002: uvm_test_top.env.sb [report_phase] Total successful transactions: 3001
# UVM_INFO FIFO_scoreboard.sv(64) @ 6002: 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: 6002 ns  Iteration: 61  Instance: /FIFO_top
```

2) Write only sequence:

⇨ *Write operation occurs successfully and wr_ack = 1:*

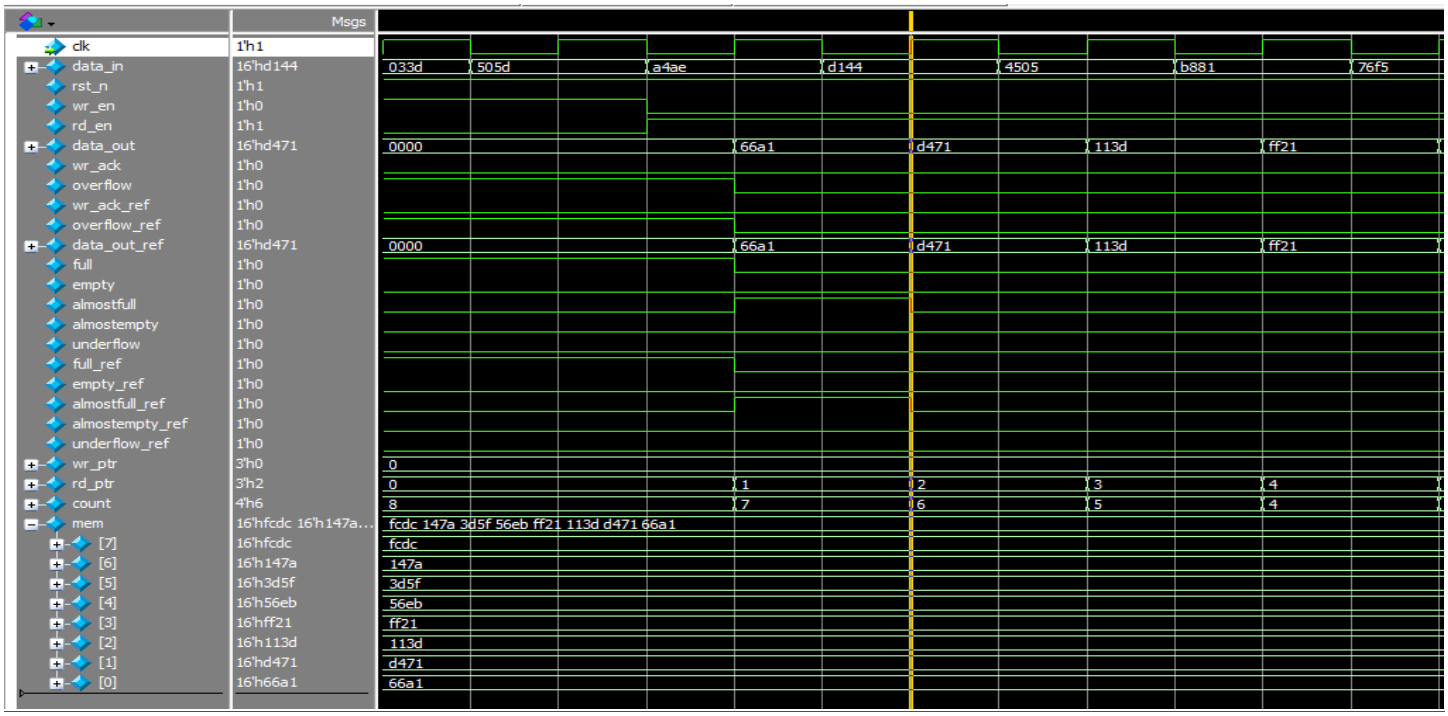


⇨ *Full and overflow are high:*

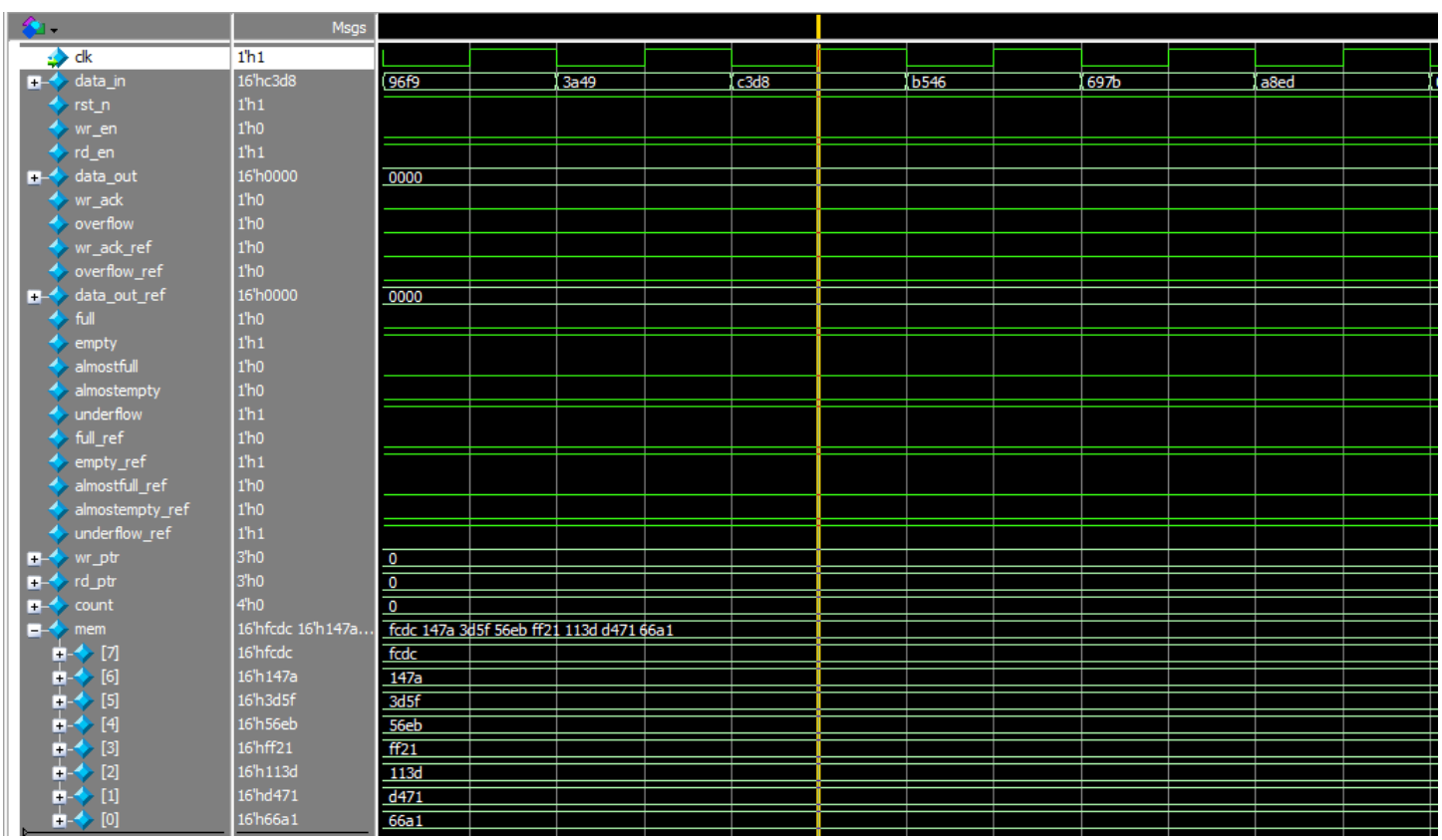


3) Read only sequence:

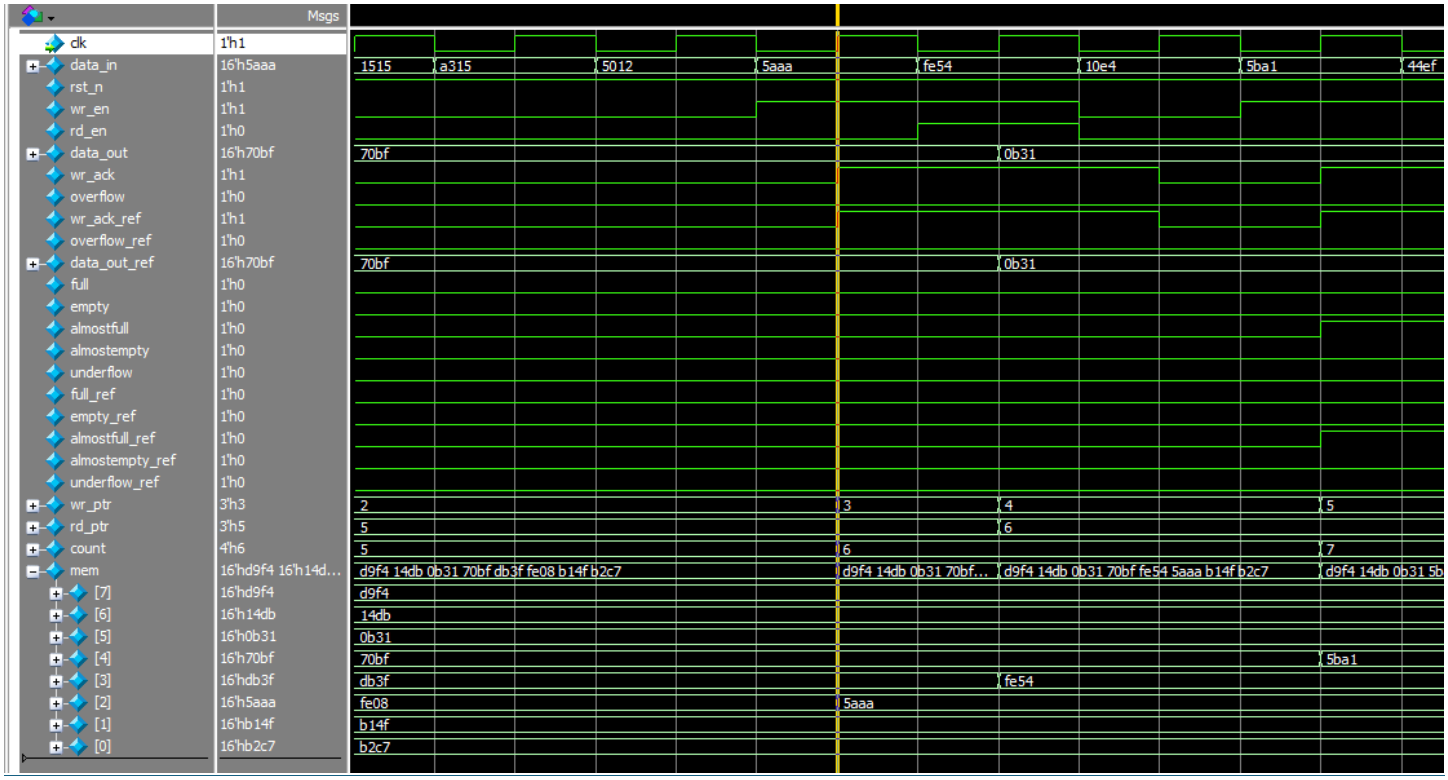
⇨ *Read operation occurs successfully:*

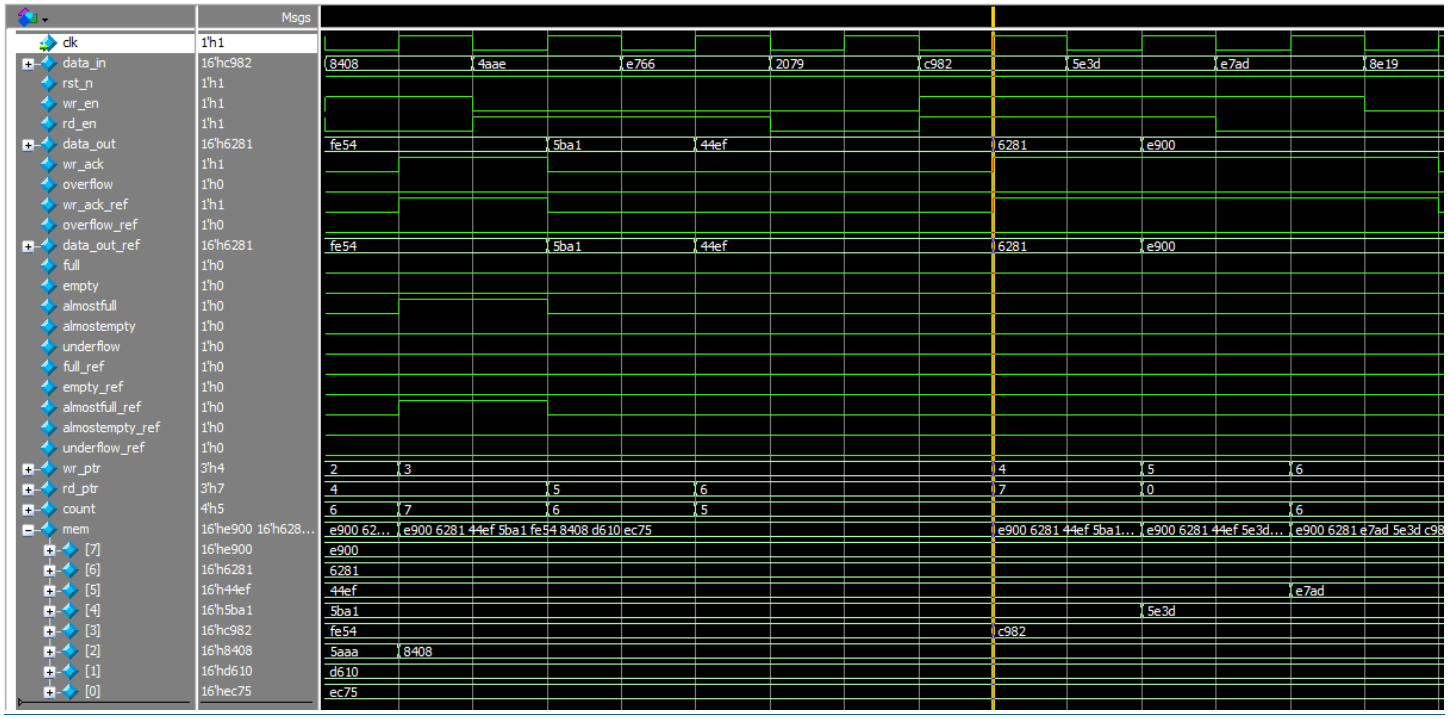


⇨ *Empty and underflow are high:*



4) Write read sequence:





⇨ ***Code coverage:***

1) Statement:

Statements - by instance (/FIFO_top/FIFO_DUT)

FIFO.sv

```
20 always @(posedge FIFO_if.clk or negedge FIFO_if.rst_n) begin
22 wr_ptr <= 0;
23 FIFO_if.overflow <= 0;  //fix: overflow signal should be zero at reset
24 FIFO_if.wr_ack <= 0;    //fix: write_ack signal should be zero at reset
27 mem[wr_ptr] <= FIFO_if.data_in;
28 FIFO_if.wr_ack <= 1;
29 wr_ptr <= wr_ptr + 1;
30 FIFO_if.overflow <= 0;  //fix: due to FIFO is not full , so overflow should be zero
33 FIFO_if.wr_ack <= 0;
35 FIFO_if.overflow <= 1;
37 FIFO_if.overflow <= 0;
42 always @(posedge FIFO_if.clk or negedge FIFO_if.rst_n) begin
44 rd_ptr <= 0;
45 FIFO_if.underflow <= 0;   //fix: underflow signal should be zero at reset
46 FIFO_if.data_out <= 0;    //fix: dataout signal should be zero at reset
49 FIFO_if.data_out <= mem[rd_ptr];
50 rd_ptr <= rd_ptr + 1;
51 FIFO_if.underflow <= 0;    //fix: due to FIFO is not empty , so underflow should be zero
56 FIFO_if.underflow <= 1;       //fix
59 FIFO_if.underflow <= 0;        //fix
64 always @(posedge FIFO_if.clk or negedge FIFO_if.rst_n) begin
66 count <= 0;
70 count <= count-1;  //fix: when both wr_en and rd_en are high , and full=1 , only read operation will occur
73 count <= count+1; //fix: when both wr_en and rd_en are high , and empty=1 , only write operation will occur
76 count <= count; //fix: when both wr_en and rd_en are high , and both empty=0 and full=0 , both operations (read,write) will occur
79 count <= count + 1;
81 count <= count - 1;
85 assign FIFO_if.full = (count == FIFO_DEPTH)? 1 : 0;
86 assign FIFO_if.empty = (count == 0)? 1 : 0;
87 assign FIFO_if.almostfull = (count == FIFO_DEPTH-1)? 1 : 0; //fix : almostfull signal is high when count=FIFO_DEPTH-1 not FIFO_DEPTH-2
88 assign FIFO_if.almostempty = (count == 1)? 1 : 0;
```

FIFO_top.sv

```
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                      31        31         0   100.00%
```

2) Branch:

```
FIFO.sv
    21 if (!FIFO_if.rst_n) begin
    26 else if (FIFO_if.wr_en && count < FIFO_DEPTH) begin
    32 else begin
    34 if (FIFO_if.full && FIFO_if.wr_en)
    36 else
    43 if (!FIFO_if.rst_n) begin
    48 else if (FIFO_if.rd_en && count != 0) begin
    54 else begin   //fix : underflow output is sequential output not combinational
    55 if (FIFO_if.rd_en && FIFO_if.empty) begin
    58 else begin
    65 if (!FIFO_if.rst_n) begin
    68 else begin
    69 if (({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b11) && FIFO_if.full) begin
    72 else if (({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b11) && FIFO_if.empty) begin  //fix
    75 else if (({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b11) && !FIFO_if.full && !FIFO_if.empty) begin  //fix
    78 else if ( ({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b10) && !FIFO_if.full)
    80 else if ( ({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b01) && !FIFO_if.empty)
    85 assign FIFO_if.full = (count == FIFO_DEPTH)? 1 : 0;
    86 assign FIFO_if.empty = (count == 0)? 1 : 0;
    87 assign FIFO_if.almostfull = (count == FIFO_DEPTH-1)? 1 : 0; //fix : almostfull signal is high when count=FIFO_DEPTH-1 not FIFO_DEPTH-2
    88 assign FIFO_if.almostempty = (count == 1)? 1 : 0;
FIFO_top.sv
```

```
=== Instance: /FIFO_top/FIFO_DUT
=== Design Unit: work.FIFO
================================================================================
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        26        26         0   100.00%
```

3) Toggle:

```
sim:/FIFO_top/FIFO_if
    almostempty
    almostempty_ref
    almostfull
    almostfull_ref
    clk
    data_in
    data_out
    data_out_ref
    empty
    empty_ref
    full
    full_ref
    overflow
    overflow_ref
    rd_en
    rst_n
    underflow
    underflow_ref
    wr_ack
    wr_ack_ref
    wr_en
```

```
Toggles - by instance (/FIFO_top/FIFO_DUT)
sim:/FIFO_top/FIFO_DUT
    count
    rd_ptr
    wr_ptr
```

```
=== Instance: /FIFO_top/FIFO_if
=== Design Unit: work.FIFO_interface
================================================================================
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                        132       132         0   100.00%
```

```
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         20        20         0   100.00%

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

Toggle Coverage for instance /FIFO_top/FIFO_DUT --

                                    Node      1H->0L      0L->1H  "Coverage"
                                    ----------------------------------------
                                 count[3-0]        1           1      100.00
                                rd_ptr[2-0]        1           1      100.00
                                wr_ptr[2-0]        1           1      100.00
```

⇨ ***Functional coverage:***

Covergroups

Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment
/FIFO_coverage_pkg/FIFO_coverage		100.00%							
TYPE write_read_cover		100.00%	100	100.00...		✓	auto(1)		
CVP write_read_cover::rst_cvg		100.00%	100	100.00...		✓			
CVP write_read_cover::write_enable_cvg		100.00%	100	100.00...		✓			
CVP write_read_cover::read_enable_cvg		100.00%	100	100.00...		✓			
CVP write_read_cover::full_cvg		100.00%	100	100.00...		✓			
CVP write_read_cover::empty_cvg		100.00%	100	100.00...		✓			
CVP write_read_cover::almost_full_cvg		100.00%	100	100.00...		✓			
CVP write_read_cover::almost_empty_cvg		100.00%	100	100.00...		✓			
CVP write_read_cover::write_ack_cvg		100.00%	100	100.00...		✓			
CVP write_read_cover::overflow_cvg		100.00%	100	100.00...		✓			
CVP write_read_cover::underflow_cvg		100.00%	100	100.00...		✓			
CROSS write_read_cover::write_read_full		100.00%	100	100.00...		✓			
CROSS write_read_cover::write_read_empty		100.00%	100	100.00...		✓			
CROSS write_read_cover::write_read_almost_full		100.00%	100	100.00...		✓			
CROSS write_read_cover::write_read_almostempty		100.00%	100	100.00...		✓			
CROSS write_read_cover::write_read_wr_ack		100.00%	100	100.00...		✓			
CROSS write_read_cover::write_read_overflow		100.00%	100	100.00...		✓			
CROSS write_read_cover::write_read_underflow		100.00%	100	100.00...		✓			

⇨ ***Assertions:***

/FIFO_read_write_sequence_pkg::FIFO_read_write_sequence::body/#ublk#25493047#17/immed__26	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/FIFO_write_only_sequence_pkg::FIFO_write_only_sequence::body/#ublk#39281767#17/immed__26	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/FIFO_read_only_sequence_pkg::FIFO_read_only_sequence::body/#ublk#18055399#17/immed__26	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/reset_ass	Immediate	SVA	on	0	1	-	-	-	-		off	assert (!FIFO_DUT.count&!FIFO_D...	✓
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/full_ass	Immediate	SVA	on	0	1	-	-	-	-		off	assert (FIFO_if.full==(FIFO_DUT.c...	✓
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/empty_ass	Immediate	SVA	on	0	1	-	-	-	-		off	assert (FIFO_if.empty==(FIFO_DU...	✓
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/almostfull_ass	Immediate	SVA	on	0	1	-	-	-	-		off	assert (FIFO_if.almostfull==(FIFO_...	✓
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/almost_empty_ass	Immediate	SVA	on	0	1	-	-	-	-		off	assert (FIFO_if.almostempty==(FI...	✓
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/full_inactive_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.ck) dis...	✓
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/full_after_almostfull_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.ck) dis...	✓
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/almostfull_from_full_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.ck) dis...	✓
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/almostfull_inactive_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.ck) dis...	✓
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/empty_inactive_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.ck) dis...	✓
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/empty_from_almostempty_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.ck) dis...	✓
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/almsotempty_inactive_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.ck) dis...	✓
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/almsotempty_from_empty_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.ck) dis...	✓
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/overflow_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.ck) dis...	✓
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/underflow_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.ck) dis...	✓
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/wr_ack_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.ck) dis...	✓
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/wr_ack_inactive_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.ck) dis...	✓
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/wr_ptr_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.ck) dis...	✓
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/rd_ptr_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.ck) dis...	✓
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/count_inc_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.ck) dis...	✓
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/count_dec_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.ck) dis...	✓
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/count_const_assert	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge FIFO_if.ck) dis...	✓

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/full_inactive_cover	SVA	✓	Off	5	1	Unlimi...	1	100%		✓	0	0	0 ns	0
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/full_after_almostfull_cover	SVA	✓	Off	33	1	Unlimi...	1	100%		✓	0	0	0 ns	0
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/almostfull_from_full_cover	SVA	✓	Off	5	1	Unlimi...	1	100%		✓	0	0	0 ns	0
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/almostfull_inactive_cover	SVA	✓	Off	9	1	Unlimi...	1	100%		✓	0	0	0 ns	0
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/empty_inactive_cover	SVA	✓	Off	129	1	Unlimi...	1	100%		✓	0	0	0 ns	0
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/empty_from_almostempty_cover	SVA	✓	Off	62	1	Unlimi...	1	100%		✓	0	0	0 ns	0
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/almsotempty_inactive_cover	SVA	✓	Off	101	1	Unlimi...	1	100%		✓	0	0	0 ns	0
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/almsotempty_from_empty_cover	SVA	✓	Off	89	1	Unlimi...	1	100%		✓	0	0	0 ns	0
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/overflow_cover	SVA	✓	Off	674	1	Unlimi...	1	100%		✓	0	0	0 ns	0
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/underflow_cover	SVA	✓	Off	991	1	Unlimi...	1	100%		✓	0	0	0 ns	0
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/wr_ack_cover	SVA	✓	Off	705	1	Unlimi...	1	100%		✓	0	0	0 ns	0
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/wr_ack_inactive_cover	SVA	✓	Off	2054	1	Unlimi...	1	100%		✓	0	0	0 ns	0
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/wr_ptr_cover	SVA	✓	Off	705	1	Unlimi...	1	100%		✓	0	0	0 ns	0
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/rd_ptr_cover	SVA	✓	Off	388	1	Unlimi...	1	100%		✓	0	0	0 ns	0
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/count_inc_cover	SVA	✓	Off	471	1	Unlimi...	1	100%		✓	0	0	0 ns	0
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/count_dec_cover	SVA	✓	Off	192	1	Unlimi...	1	100%		✓	0	0	0 ns	0
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/count_const_cover	SVA	✓	Off	194	1	Unlimi...	1	100%		✓	0	0	0 ns	0

```
Assertion Coverage:
    Assertions                      22         22          0   100.00%
----------------------------------------------------------------------
Name                  File(Line)                    Failure      Pass
                                                    Count        Count
----------------------------------------------------------------------
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/reset_ass
                      FIFO_SVA.sv(10)                     0            1
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/full_ass
                      FIFO_SVA.sv(13)                     0            1
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/empty_ass
                      FIFO_SVA.sv(14)                     0            1
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/almostfull_ass
                      FIFO_SVA.sv(15)                     0            1
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/almost_empty_ass
                      FIFO_SVA.sv(16)                     0            1
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/full_inactive_assert
                      FIFO_SVA.sv(101)                    0            1
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/full_after_almostfull_assert
                      FIFO_SVA.sv(102)                    0            1
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/almostfull_from_full_assert
                      FIFO_SVA.sv(103)                    0            1
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/almostfull_inactive_assert
                      FIFO_SVA.sv(104)                    0            1
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/empty_inactive_assert
                      FIFO_SVA.sv(105)                    0            1
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/empty_from_almostempty_assert
                      FIFO_SVA.sv(106)                    0            1
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/almsotempty_inactive_assert
                      FIFO_SVA.sv(107)                    0            1
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/almsotempty_from_empty_assert
                      FIFO_SVA.sv(108)                    0            1
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/overflow_assert
                      FIFO_SVA.sv(109)                    0            1
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/underflow_assert
                      FIFO_SVA.sv(110)                    0            1
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/wr_ack_assert
                      FIFO_SVA.sv(111)                    0            1
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/wr_ack_inactive_assert
                      FIFO_SVA.sv(112)                    0            1
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/wr_ptr_assert
                      FIFO_SVA.sv(113)                    0            1
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/rd_ptr_assert
                      FIFO_SVA.sv(114)                    0            1
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/count_inc_assert
                      FIFO_SVA.sv(115)                    0            1
/FIFO_top/FIFO_DUT/FIFO_SVA_INST/count_dec_assert
                      FIFO_SVA.sv(116)                    0            1
```