FIFO

UVM Project

By: Mohammed Anwar Abd allatif

Under supervision of Eng. Kareem Waseem

Design has 6 bugs:

1. Reset signals overflow, wr_ack and underflow (data_out not to be included)
2. Unhandled 2 cases:

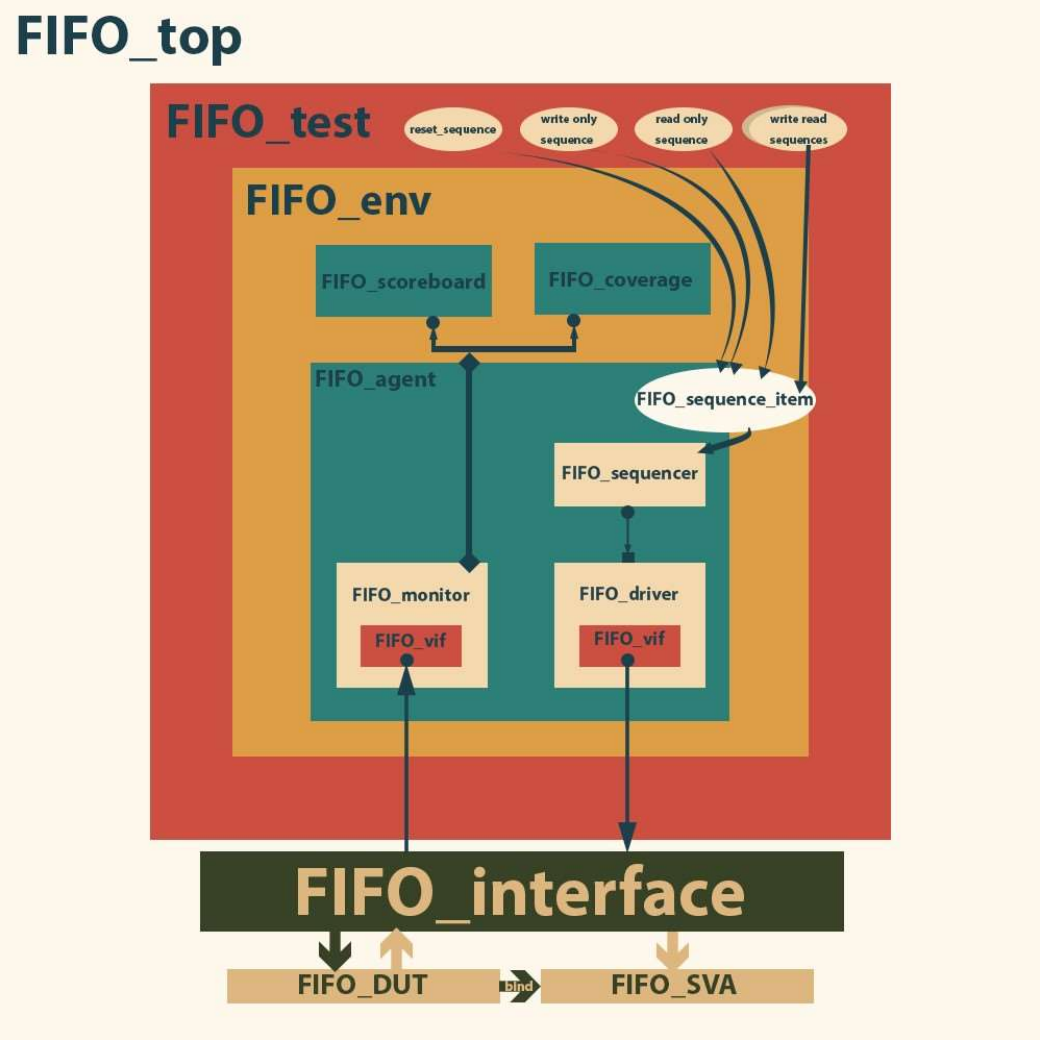
- If a read and write enables were high and the FIFO was empty, only writing will take place.
- If a read and write enables were high and the FIFO was full, only reading will take place.

3. underflow is sequential not combinational.
4. almostfull flag: FIFO_DEPTH-2 corrected to FIFO_DEPTH-1.
5. When successful write operation has occured, overflow can't be high.
6. When successful read operation has occured, underflow can't be high.

Verification plan:

Label	Description	Stimulus Generation	Functional Coverage	Functionality Check
LABEL1	When the rst_n is asserted, All flags & internal signals should equal 0	Randomized less often 95 off & 5% on during the simulation	.	Immediate assertion to check async rst_n functionality as it is async
LABEL2	When rst_n is deactivated & the FIFO is full of elements (conut=depth), The full flag should be high	Randomized, Write enable to be high with distribution of the value WR_EN_ON_DIST and to be low with 100-WR_EN_ON_DISTduring the simulation	Cross coverage: WRITE_READ_FULL_CROSS, covers write & read enables and full flag	Immediate assertion to check full flag functionality as it is combinational
LABEL3	When rst_n is deactivated & the FIFO has one element left(conut=depth-1), The almostfull flag should be high	Randomized, Write enable to be high with distribution of the value WR_EN_ON_DIST and to be low with 100-WR_EN_ON_DISTduring the simulation	Cross coverage: WRITE_READ_ALMOST_FULL_CROSS, covers write & read enables and almostfull flag	Immediate assertion to check almostfull flag functionality as it is combinational
LABEL4	When rst_n is deactivated & the FIFO has no element inside (conut=0), The empty flag should be high	Randomized, Read enable to be high with distribution of the value RD_EN_ON_DIST and to be low with 100-RD_EN_ON_DIST during the simulation	Cross coverage: WRITE_READ_EMPTY_CROSS, covers write & read enables and empty flag	Immediate assertion to check empty flag functionality as it is combinational
LABEL5	When rst_n is deactivated & the FIFO has only one element inside (conut=1), The empty flag should be high	Randomized, Read enable to be high with distribution of the value RD_EN_ON_DIST and to be low with 100-RD_EN_ON_DIST during the simulation	Cross coverage: WRITE_READ_ALMSOT_EMPTY_CROSS, covers write & read enables and almostempty flag	Immediate assertion to check almostempty flag functionality as it is combinational
LABEL6	When rst_n is deactivated, You want to read & the FIFO is not full, The wr_ack flag should be high and wr_ptr should increment	Randomized, Write enable to be high with distribution of the value WR_EN_ON_DIST and to be low with 100-WR_EN_ON_DISTduring the simulation	Cross coverage: WRITE_READ_WR_ACK_CROSS, covers write & read enables and wr_ack flag	Concurrent assertion to check wr_ptr & wr_ack flag functionality as they are sequential
LABEL7	When rst_n is deactivated, You want to read & the FIFO is full, The overflow flag should be high	Randomized, Write enable to be high with distribution of the value WR_EN_ON_DIST and to be low with 100-WR_EN_ON_DISTduring the simulation	Cross coverage: WRITE_READ_OVERFLOW_CROSS, covers write & read enables and overflow flag	Concurrent assertion to check overflow flag functionality as it is sequential
LABEL8	When rst_n is deactivated, You want to read & the FIFO is not empty rd_ptr should increment and data_out should equal mem[rd_ptr]	Randomized, Read enable to be high with distribution of the value RD_EN_ON_DIST and to be low with 100-RD_EN_ON_DIST during the simulation	.	Concurrent assertion to check rd_ptr functionality as it is sequential, data_out checked against refernce model to check functionality
LABEL9	When rst_n is deactivated, You want to read & the FIFO is empty underflow flag should be high	Randomized, Read enable to be high with distribution of the value RD_EN_ON_DIST and to be low with 100-RD_EN_ON_DIST during the	Cross coverage: WRITE_READ_UNDERFLOW_CROSS, covers write & read enables and overflow flag	Concurrent assertion to check underflow flag functionality as it is sequential

UVM testbench structure:



Testbench flow

- FIFO_top module:
  1. instantiates the DUT, FIFO_interface & bind assertions (FIFO_SVA).
  2. Generate the clock.
  3. Passes interface (virtual FIFO_interface) using configuration database (Shared database between components).
  4. Runs test.
- FIFO_test:
  1. Build the FIFO_env & Sequences.

2. Retrieve the virtual interface from the configuration database by configuration object (Object holds configuration settings and parameters for UVM components).
3. Sets the configuration object into the configuration database.
4. Builds the environment (FIFO_env) 5. Starts sequences on the sequencer.

- Sequences:
  1. There are 5 sequences: Reset, Write only, Read only, Write Read & write read empty sequences.
  2. Core stimulus of any verification plan.
  3. Written within task body.
- FIFO_sequence_item:
  1. Data fields to communicate with DUT (Input & Output signals).
  2. Randomizes signals.
  3. Constraints blocks are added here to ensure verification plan. FIFO_env:

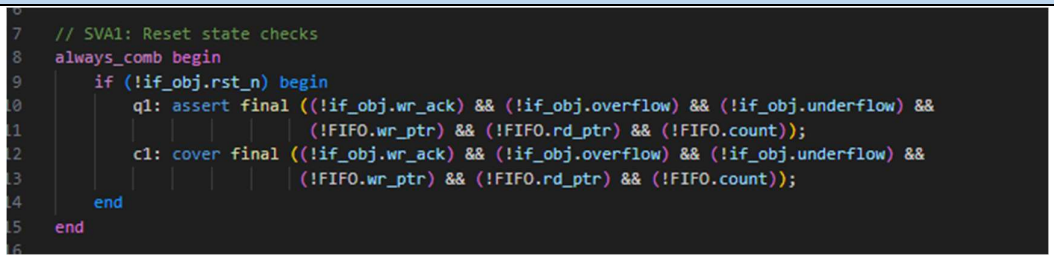
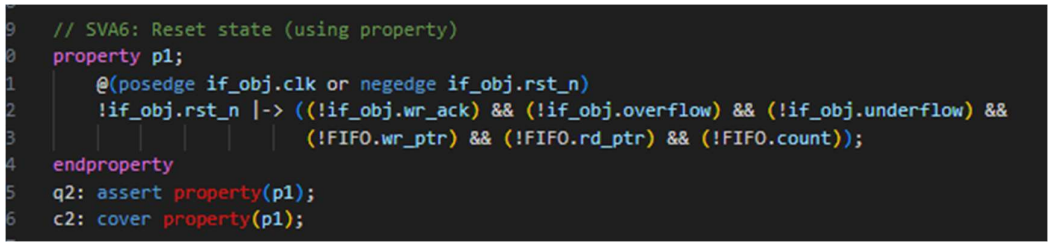
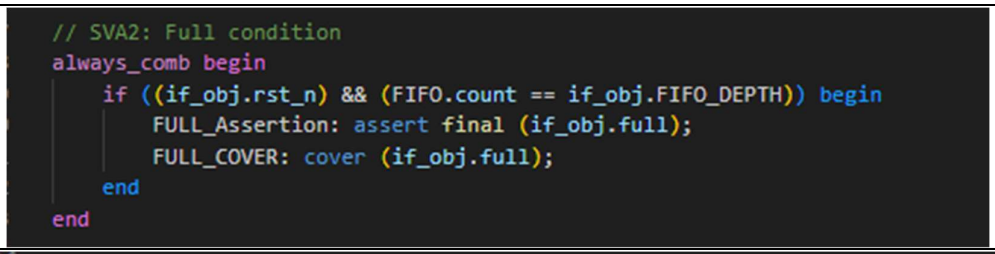
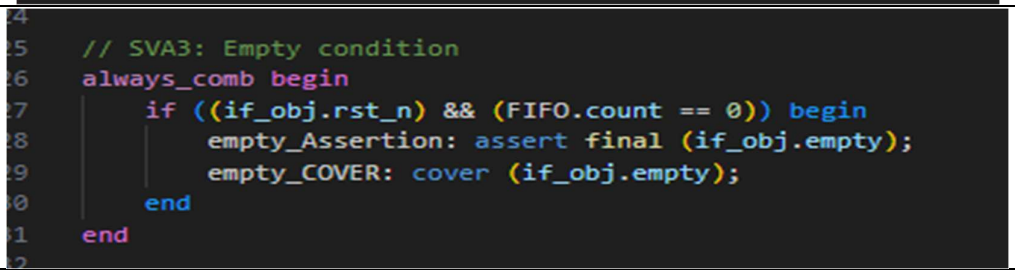
  Builds and connects scoreboard (FIFO_scoreboard), Coverage collector (FIFO_coverage), agent (FIFO_agent) and analysis components (Ports & Exports).
- FIFO_sequencer:
  Generates transactions as class objects and sends it to the driver (FIFO_driver) for execution.

- FIFO_driver:
  1. Pulls the next item from the sequencer.
  2. Drives the sequence item in the run_phase task using the virtual interface.
- FIFO_monitor:
  Captures signals information from DUT, translates it into sequence items and finally sends it analysis components (Ports & Exports).
- FIFO_scoreboard:
  1. Receives sequence items from the monitor.

2. Runs input signals to the reference model (Task or Module but I have used a task) to compare the DUT output with the expected output to check the functionality of the FIFO.

- FIFO_coverage:
  1. Receives sequence items from the monitor.
  2. Contains the covergroups to ensure the verification plan.
  3. Samples the data fields for functional coverage.

2 Assertions table:

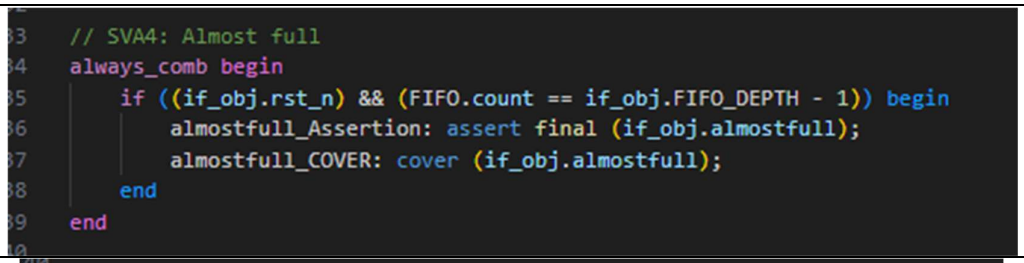
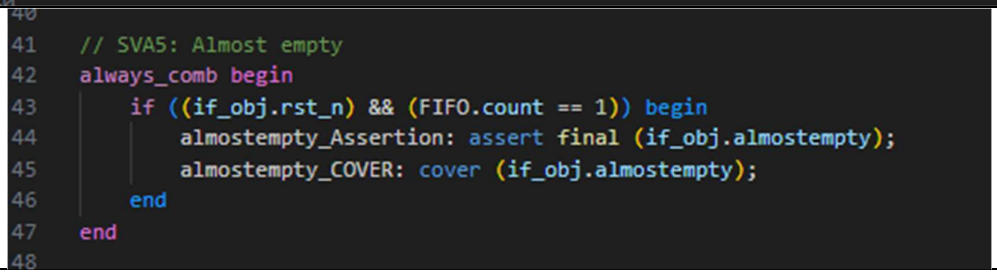
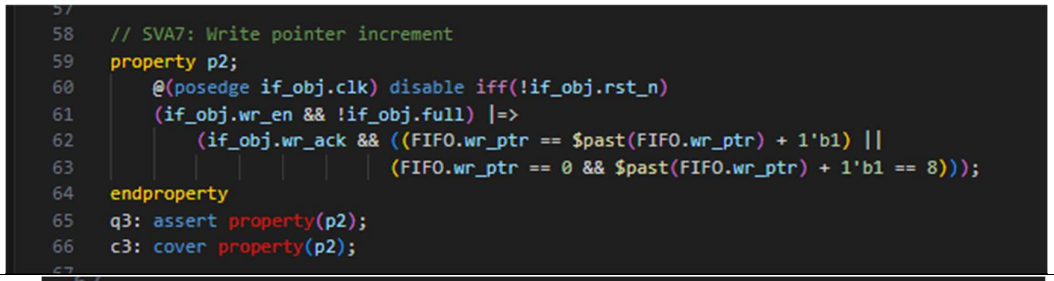
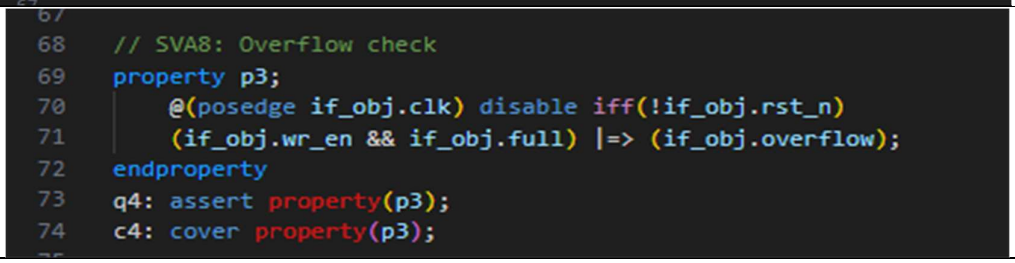
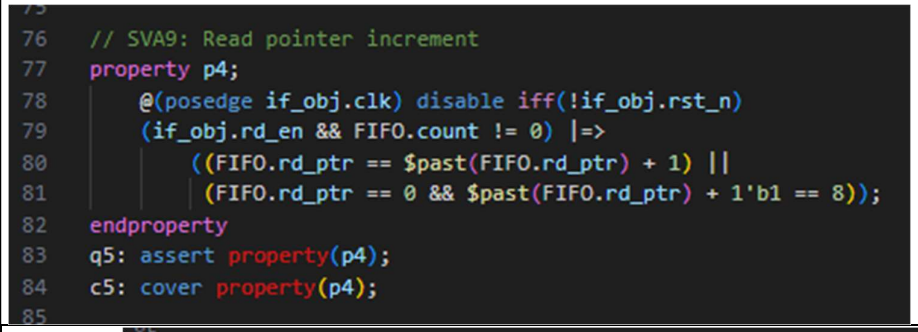
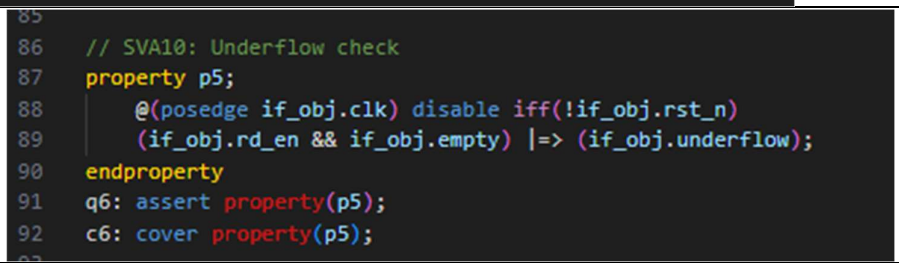
Feature	Assertion
Whenever the rst_n is active, All sequential flags and internal signals should be low.	 
Whenever the rst_n is deactivated & number of FIFO elements equal FIFO maximum depth, full flag should be high.	
Whenever the rst_n is deactivated & number of FIFO elements equal zero, empty flag should be high.	

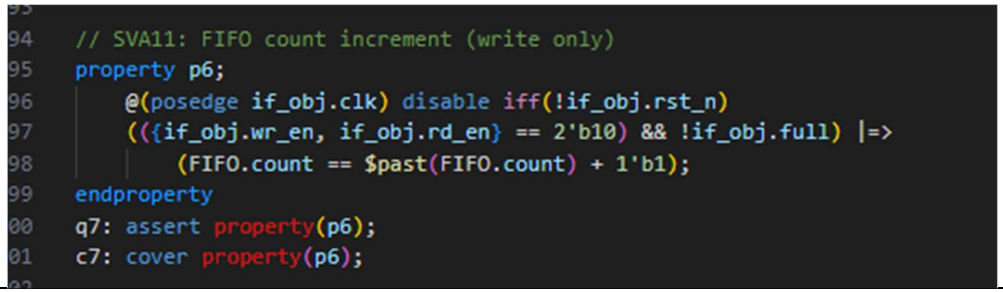
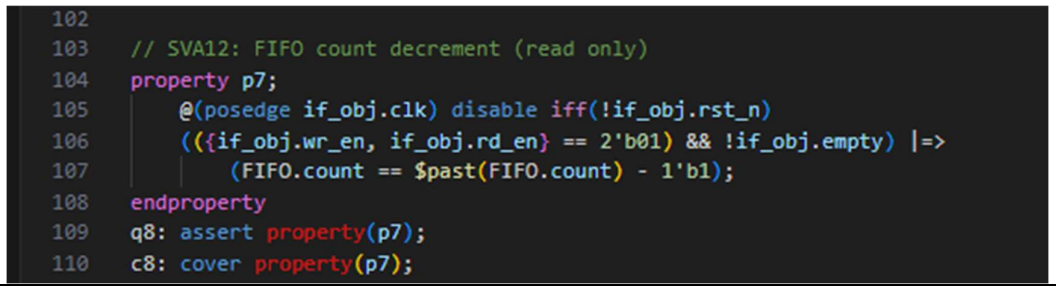
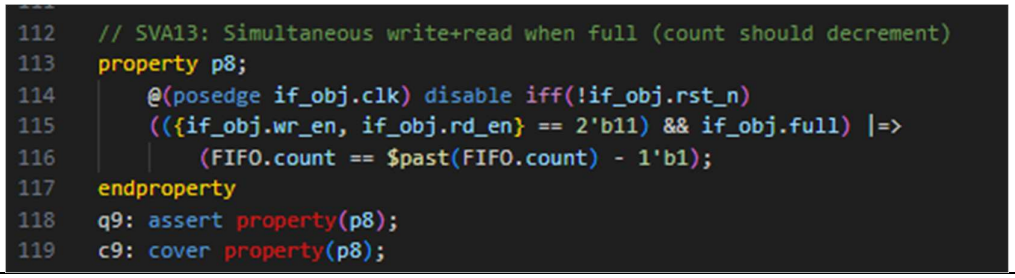
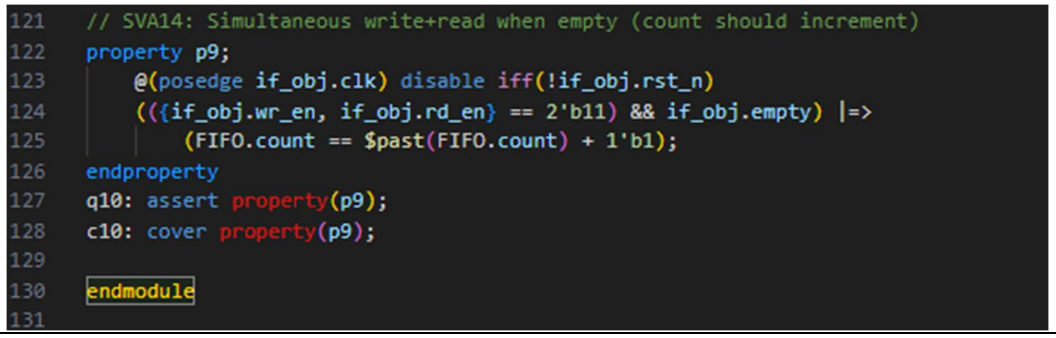
```
// SVA1: Reset state checks
always_comb begin
    if (!if_obj.rst_n) begin
        q1: assert final ((!if_obj.wr_ack) && (!if_obj.overflow) && (!if_obj.underflow) &&
                          (!FIFO.wr_ptr) && (!FIFO.rd_ptr) && (!FIFO.count));
        c1: cover final ((!if_obj.wr_ack) && (!if_obj.overflow) && (!if_obj.underflow) &&
                         (!FIFO.wr_ptr) && (!FIFO.rd_ptr) && (!FIFO.count));
    end
end
```

```
// SVA6: Reset state (using property)
property p1;
    @(posedge if_obj.clk or negedge if_obj.rst_n)
    !if_obj.rst_n |-> ((!if_obj.wr_ack) && (!if_obj.overflow) && (!if_obj.underflow) &&
                       (!FIFO.wr_ptr) && (!FIFO.rd_ptr) && (!FIFO.count));
endproperty
q2: assert property(p1);
c2: cover property(p1);
```

```
// SVA2: Full condition
always_comb begin
    if ((if_obj.rst_n) && (FIFO.count == if_obj.FIFO_DEPTH)) begin
        FULL_Assertion: assert final (if_obj.full);
        FULL_COVER: cover (if_obj.full);
    end
end
```

```
// SVA3: Empty condition
always_comb begin
    if ((if_obj.rst_n) && (FIFO.count == 0)) begin
        empty_Assertion: assert final (if_obj.empty);
        empty_COVER: cover (if_obj.empty);
    end
end
```

Whenever the rst_n is deactivated & number of FIFO elements equal FIFO maximum depth -1, almostfull flag should be high.	
Whenever the rst_n is deactivated & number of FIFO elements equal 1, almostempty flag should be high.	
Whenever the rst_n is deactivated, Write enable is high & FIFO is not full, wr_ack should be high & wr_ptr should increment.	
Whenever the rst_n is deactivated, Write enable is high & FIFO is full, overflow should be high.	
Whenever the rst_n is deactivated, Read enable is high & number of FIFO elements doesn't equal zero, rd_ptr should increment.	
Whenever the rst_n is deactivated, Read enable is high & FIFO is empty, underflow should be high.	

Whenever the rst_n is deactivated, Write enable is high & FIFO is not full, Write operation should take place.	
Whenever the rst_n is deactivated, read enable is high & FIFO is not empty, Read operation should take place.	
Whenever the rst_n is deactivated, Both of read & write enables are high & FIFO is empty, Write operation should take place.	
Whenever the rst_n is deactivated, Both of read & write enables are high & FIFO is not full, Read operation should take place.	

```
94   // SVA11: FIFO count increment (write only)
95   property p6;
96       @(posedge if_obj.clk) disable iff(!if_obj.rst_n)
97       (({if_obj.wr_en, if_obj.rd_en} == 2'b10) && !if_obj.full) |=>
98           (FIFO.count == $past(FIFO.count) + 1'b1);
99   endproperty
00   q7: assert property(p6);
01   c7: cover property(p6);
```

```
102
103   // SVA12: FIFO count decrement (read only)
104   property p7;
105       @(posedge if_obj.clk) disable iff(!if_obj.rst_n)
106       (({if_obj.wr_en, if_obj.rd_en} == 2'b01) && !if_obj.empty) |=>
107           (FIFO.count == $past(FIFO.count) - 1'b1);
108   endproperty
109   q8: assert property(p7);
110   c8: cover property(p7);
```

```
112   // SVA13: Simultaneous write+read when full (count should decrement)
113   property p8;
114       @(posedge if_obj.clk) disable iff(!if_obj.rst_n)
115       (({if_obj.wr_en, if_obj.rd_en} == 2'b11) && if_obj.full) |=>
116           (FIFO.count == $past(FIFO.count) - 1'b1);
117   endproperty
118   q9: assert property(p8);
119   c9: cover property(p8);
```

```
121   // SVA14: Simultaneous write+read when empty (count should increment)
122   property p9;
123       @(posedge if_obj.clk) disable iff(!if_obj.rst_n)
124       (({if_obj.wr_en, if_obj.rd_en} == 2'b11) && if_obj.empty) |=>
125           (FIFO.count == $past(FIFO.count) + 1'b1);
126   endproperty
127   q10: assert property(p9);
128   c10: cover property(p9);
129
130   endmodule
131
```

3 Design code before debug :

D: > Digital Verfication debloma > Projects > Project2(FIFO WITH UVM) > RFIO_UVM > FIFO.v

```
// Description: FIFO Design
//
//////////////////////////////////////////////////////////////////////////////////
module FIFO(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
```

Design after debug :

D: > Digital Verfication debloma > Projects > Project2(FIFO WITH UVM) > FIFO_UVM > FIFO_after.sv

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


// declaration of Memory (FIFO)
reg [if_obj.FIFO_WIDTH-1:0] mem [if_obj.FIFO_DEPTH-1:0];

// Declaration of read & write pointers
reg [if_obj.max_fifo_addr-1:0] wr_ptr, rd_ptr;
reg [if_obj.max_fifo_addr:0] count;             // extra bit to distinguish between full & empty flags & it represents the fill level of the FIFO

// always block specialized for writing operation
always @(posedge if_obj.clk or negedge if_obj.rst_n) begin
    if (!if_obj.rst_n) begin
        wr_ptr <= 0;
        if_obj.overflow <= 0;
        if_obj.wr_ack <= 0;           // reset the sequential outputs as wr_ack , overflow
    end
    else if (if_obj.wr_en && count < if_obj.FIFO_DEPTH) begin
        mem[wr_ptr] <= if_obj.data_in;
        if_obj.wr_ack <= 1;
        wr_ptr <= wr_ptr + 1;
        if_obj.overflow <=0 ;
    end
    else begin
        if_obj.wr_ack <= 0;
        if (if_obj.full && if_obj.wr_en)
            if_obj.overflow <= 1;
        else
            if_obj.overflow <= 0;
    end
end

// always block specialized for reading operation
always @(posedge if_obj.clk or negedge if_obj.rst_n) begin
    if (!if_obj.rst_n) begin
        rd_ptr <= 0;
        if_obj.underflow <= 0;

    end
    else if (if_obj.rd_en && count != 0) begin
        if_obj.data_out <= mem[rd_ptr];
        rd_ptr <= rd_ptr + 1;
        if_obj.underflow <=0;
        end
    else
    begin
        if(if_obj.empty && if_obj.rd_en)
            if_obj.underflow  <= 1;
        else
            if_obj.underflow  <= 0;
    end
end

```

```
54  // always block specialized for counter signal
55  always @(posedge if_obj.clk or negedge if_obj.rst_n) begin
56      if (!if_obj.rst_n) begin
57          count <= 0;
58      end
59      else begin
60          if  ( ({if_obj.wr_en, if_obj.rd_en} == 2'b10) && !if_obj.full)
61              count <= count + 1;
62          else if ( ({if_obj.wr_en, if_obj.rd_en} == 2'b01) && !if_obj.empty)
63              count <= count - 1;
64          else if (({if_obj.wr_en, if_obj.rd_en} == 2'b11) && if_obj.full)      // priority for write operation
65              count <= count - 1;
66          else if (({if_obj.wr_en, if_obj.rd_en} == 2'b11) && if_obj.empty)      // priority for read operation
67              count <= count + 1;
68      end
69  end
70
71  // continous assignment for the combinational outputs
72  assign if_obj.full = (count == if_obj.FIFO_DEPTH)? 1 : 0;
73  assign if_obj.empty = (count == 0)? 1 : 0;
74  assign if_obj.almostfull = (count == if_obj.FIFO_DEPTH-1)? 1 : 0;
75  assign if_obj.almostempty = (count == 1)? 1 : 0;
76
77  endmodule
```

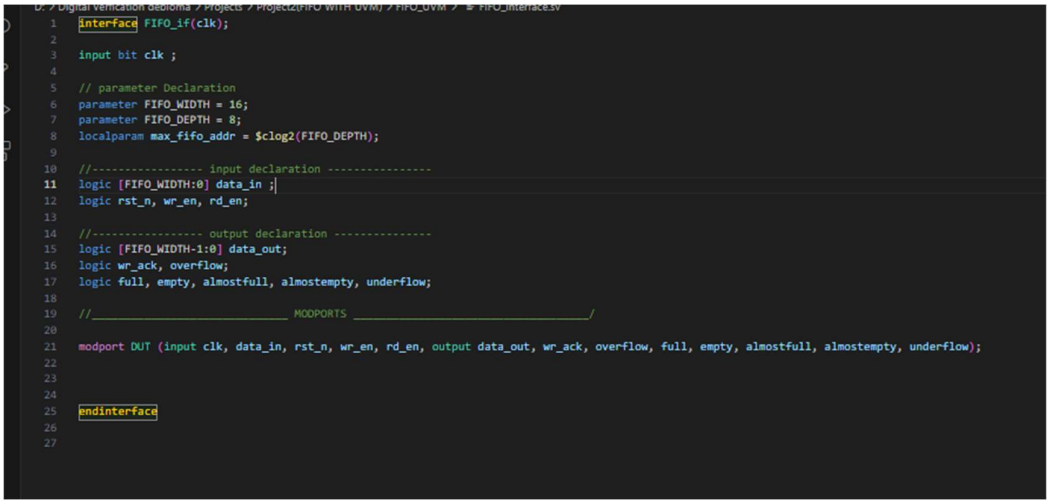
FIFO_SVA(assertion) :

```
1   module FIFO_SVA(FIFO_if.DUT if_obj);
2
3   //=======================================================
4   // Assertions + Cover Properties
5   //=======================================================
6
7   // SVA1: Reset state checks
8   always_comb begin
9       if (!if_obj.rst_n) begin
10          q1: assert final ((!if_obj.wr_ack) && (!if_obj.overflow) && (!if_obj.underflow) &&
11                            (!FIFO.wr_ptr) && (!FIFO.rd_ptr) && (!FIFO.count));
12          c1: cover final ((!if_obj.wr_ack) && (!if_obj.overflow) && (!if_obj.underflow) &&
13                           (!FIFO.wr_ptr) && (!FIFO.rd_ptr) && (!FIFO.count));
14      end
15  end
16
17  // SVA2: Full condition
18  always_comb begin
19      if ((if_obj.rst_n) && (FIFO.count == if_obj.FIFO_DEPTH)) begin
20          FULL_Assertion: assert final (if_obj.full);
21          FULL_COVER: cover (if_obj.full);
22      end
23  end
24
25  // SVA3: Empty condition
26  always_comb begin
27      if ((if_obj.rst_n) && (FIFO.count == 0)) begin
28          empty_Assertion: assert final (if_obj.empty);
29          empty_COVER: cover (if_obj.empty);
30      end
31  end
32
33  // SVA4: Almost full
34  always_comb begin
35      if ((if_obj.rst_n) && (FIFO.count == if_obj.FIFO_DEPTH - 1)) begin
36          almostfull_Assertion: assert final (if_obj.almostfull);
37          almostfull_COVER: cover (if_obj.almostfull);
38      end
39  end
40
41  // SVA5: Almost empty
42  always_comb begin
43      if ((if_obj.rst_n) && (FIFO.count == 1)) begin
44          almostempty_Assertion: assert final (if_obj.almostempty);
45          almostempty_COVER: cover (if_obj.almostempty);
46      end
47  end
48
49  // SVA6: Reset state (using property)
50  property p1;
51      @(posedge if_obj.clk or negedge if_obj.rst_n)
52      !if_obj.rst_n |-> ((!if_obj.wr_ack) && (!if_obj.overflow) && (!if_obj.underflow) &&
53                         (!FIFO.wr_ptr) && (!FIFO.rd_ptr) && (!FIFO.count));
54  endproperty
55  q2: assert property(p1);
56  c2: cover property(p1);
```

```
58  // SVA7: Write pointer increment
59  property p2;
60      @(posedge if_obj.clk) disable iff(!if_obj.rst_n)
61      (if_obj.wr_en && !if_obj.full) |=>
62          (if_obj.wr_ack && ((FIFO.wr_ptr == $past(FIFO.wr_ptr) + 1'b1) ||
63                             (FIFO.wr_ptr == 0 && $past(FIFO.wr_ptr) + 1'b1 == 8)));
64  endproperty
65  q3: assert property(p2);
66  c3: cover property(p2);
67
68  // SVA8: Overflow check
69  property p3;
70      @(posedge if_obj.clk) disable iff(!if_obj.rst_n)
71      (if_obj.wr_en && if_obj.full) |=> (if_obj.overflow);
72  endproperty
73  q4: assert property(p3);
74  c4: cover property(p3);
75
76  // SVA9: Read pointer increment
77  property p4;
78      @(posedge if_obj.clk) disable iff(!if_obj.rst_n)
79      (if_obj.rd_en && FIFO.count != 0) |=>
80          ((FIFO.rd_ptr == $past(FIFO.rd_ptr) + 1) ||
81          (FIFO.rd_ptr == 0 && $past(FIFO.rd_ptr) + 1'b1 == 8));
82  endproperty
83  q5: assert property(p4);
84  c5: cover property(p4);
85
86  // SVA10: Underflow check
87  property p5;
88      @(posedge if_obj.clk) disable iff(!if_obj.rst_n)
89      (if_obj.rd_en && if_obj.empty) |=> (if_obj.underflow);
90  endproperty
91  q6: assert property(p5);
92  c6: cover property(p5);
93
94  // SVA11: FIFO count increment (write only)
95  property p6;
96      @(posedge if_obj.clk) disable iff(!if_obj.rst_n)
97      (({if_obj.wr_en, if_obj.rd_en} == 2'b10) && !if_obj.full) |=>
98          (FIFO.count == $past(FIFO.count) + 1'b1);
99  endproperty
100 q7: assert property(p6);
101 c7: cover property(p6);
102
103 // SVA12: FIFO count decrement (read only)
104 property p7;
105     @(posedge if_obj.clk) disable iff(!if_obj.rst_n)
106     (({if_obj.wr_en, if_obj.rd_en} == 2'b01) && !if_obj.empty) |=>
107         (FIFO.count == $past(FIFO.count) - 1'b1);
108 endproperty
109 q8: assert property(p7);
110 c8: cover property(p7);
111
112 // SVA13: Simultaneous write+read when full (count should decrement)
113 property p8;
114     @(posedge if_obj.clk) disable iff(!if_obj.rst_n)
115     (({if_obj.wr_en, if_obj.rd_en} == 2'b11) && if_obj.full) |=>
116         (FIFO.count == $past(FIFO.count) - 1'b1);
117 endproperty
```

```
112 // SVA13: Simultaneous write+read when full (count should decrement)
113 property p8;
114     @(posedge if_obj.clk) disable iff(!if_obj.rst_n)
115     (({if_obj.wr_en, if_obj.rd_en} == 2'b11) && if_obj.full) |=>
116         (FIFO.count == $past(FIFO.count) - 1'b1);
117 endproperty
118 q9: assert property(p8);
119 c9: cover property(p8);
120
121 // SVA14: Simultaneous write+read when empty (count should increment)
122 property p9;
123     @(posedge if_obj.clk) disable iff(!if_obj.rst_n)
124     (({if_obj.wr_en, if_obj.rd_en} == 2'b11) && if_obj.empty) |=>
125         (FIFO.count == $past(FIFO.count) + 1'b1);
126 endproperty
127 q10: assert property(p9);
128 c10: cover property(p9);
129
130 endmodule
131
```

Interface:



```sv
interface FIFO_if(clk);

input bit clk ;

// parameter Declaration
parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;
localparam max_fifo_addr = $clog2(FIFO_DEPTH);

//---------------- input declaration ---------------
logic [FIFO_WIDTH:0] data_in ;
logic rst_n, wr_en, rd_en;

//---------------- output declaration --------------
logic [FIFO_WIDTH-1:0] data_out;
logic wr_ack, overflow;
logic full, empty, almostfull, almostempty, underflow;

//_____________________________ MODPORTS ____________________________________/

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



endinterface
```

FIFP_top :

D: > Digital Verfication debloma > Projects > Project2(FIFO WITH UVM) > FIFO_UVM > FIFO_top.sv

```sv
import uvm_pkg::*;
`include "uvm_macros.svh"
import FIFO_test_pkg::*;
module FIFO_top();
    bit clk;
initial begin
    clk=0;
    forever begin
        #1 clk = ~clk;
end
end
FIFO_if if_obj(clk);
FIFO DUT(if_obj);


bind FIFO FIFO_SVA SAV(if_obj);
initial begin
    uvm_config_db#(virtual FIFO_if)::set(null,"uvm_test_top","FIFO_IF",if_obj);
    run_test("FIFO_test");
end

endmodule
```

FIFO_test:

D: > Digital Verfication debloma > Projects > Project2(FIFO WITH UVM) > FIFO_UVM > FIFO_test_pkg.sv

```
package FIFO_test_pkg;
  `include "uvm_macros.svh"
  import uvm_pkg::*;
  import FIFO_env_pkg::*;
  import FIFO_sequence_pkg::*;
  import FIFO_config_obj_pkg::*;

  class FIFO_test extends uvm_test;
    `uvm_component_utils(FIFO_test);

    FIFO_env env;
    virtual FIFO_if fifo_driver_vif;
    FIFO_object fifo_config_obj_test;  // Replace object with the correct config class type
    FIFO_reset_sequence reset_seq;
    FIFO_read_only_sequence read_seq;
    FIFO_write_only_sequence write_seq;
    FIFO_read_write_sequence rd_wr_seq;
    FIFO_read_write_empty_sequence  rd_wr_em_seq;
  //_____________COSTRUCTOR_____________//
    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);

      // Replace object with ALSU_config (the correct class type for the configuration object)
      fifo_config_obj_test = FIFO_object::type_id::create("fifo_config_obj_test", this);
      // creat your five sequence

      reset_seq = FIFO_reset_sequence::type_id::create("reset_seq");
       write_seq = FIFO_write_only_sequence::type_id::create("write_seq");
      read_seq = FIFO_read_only_sequence::type_id::create("read_seq");
      rd_wr_seq = FIFO_read_write_sequence::type_id::create("rd_wr_seq");
      rd_wr_em_seq=FIFO_read_write_empty_sequence::type_id::create("rd_wr_em_seq");
      // Retrieve the virtual interface from the config object
      if (!uvm_config_db#(virtual FIFO_if)::get(this, "", "FIFO_IF", fifo_config_obj_test.fifo_config_vif)) begin
        `uvm_fatal("build_phase", "the test unable to get the virtual interface");
      end
       fifo_config_obj_test.sel_mod = UVM_ACTIVE;

      uvm_config_db #(FIFO_object)::set(this, "*", "CGO", fifo_config_obj_test);  // Set the config object in the DB
    endfunction

    /******* RUN PHASE *******/
    task run_phase(uvm_phase phase);
      super.run_phase(phase);
      phase.raise_objection(this);

      // Reset sequence
      `uvm_info("run_phase", "reset asserted", UVM_LOW);
    reset_seq.start(env.agt.sqr);
      `uvm_info("run_phase", "reset deasserted", UVM_LOW);

      // READ sequence
      `uvm_info("run_phase", "WRITE started", UVM_LOW);
      write_seq.start(env.agt.sqr);
      `uvm_info("run_phase", "WRITE ended", UVM_LOW);

      // WRITE sequence
      `uvm_info("run_phase", "READ started", UVM_LOW);
      read_seq.start(env.agt.sqr);
      `uvm_info("run_phase", "READ ended", UVM_LOW);


      //READ  WRITE sequence
      `uvm_info("run_phase", "READ WRITE started", UVM_LOW);
      rd_wr_seq.start(env.agt.sqr);
      `uvm_info("run_phase", "READ WRITE ended", UVM_LOW);


      // WRITE sequence
      `uvm_info("run_phase", "READ WRITE EMPTY started", UVM_LOW);
      rd_wr_em_seq.start(env.agt.sqr);
      `uvm_info("run_phase", "READ WRITE EMPTY ended", UVM_LOW);
     phase.drop_objection(this);

    endtask
  endclass
endpackage
```

FIFO_env :

D: > Digital Verfication debloma > Projects > Project2(FIFO WITH UVM) > FIFO_UVM > FIFO_env_pkg.sv

```
1   package FIFO_env_pkg;
2   `include "uvm_macros.svh"
3   import uvm_pkg::*;
4   import FIFO_sequence_item_pkg::*;
5   import FIFO_agent_pkg ::*;
6   import FIFO_scoreboard_pkg::*;
7   import FIFO_coverage_pkg::*;
8
9   class FIFO_env extends uvm_env;
10  `uvm_component_utils(FIFO_env);
11  FIFO_agent agt;
12  FIFO_scoreboard sb;
13  FIFO_coverage cov;
14  uvm_analysis_port #(FIFO_seq_item)agt_ap;
15
16  function  new(string name ="FIFO_env", uvm_component parent=null);
17  super.new(name,parent);
18  endfunction
19  /****************buiLed phase *************/
20  function void build_phase(uvm_phase phase);
21  super.build_phase(phase);
22  agt=FIFO_agent::type_id::create("agt",this);
23  sb=FIFO_scoreboard::type_id::create("sb",this);
24  cov=FIFO_coverage::type_id::create("cov",this);
25  agt_ap=new("agt_ap",this);
26  endfunction
27
28  /************** connect phase *************/
29    function void connect_phase(uvm_phase phase);
30              super.connect_phase(phase);
31              agt.agt_ap.connect(sb.sb_export);
32              agt.agt_ap.connect (cov.cov_export);
33          endfunction
34      endclass
35  endpackage
```

Fifo_scoreboard:

D: > Digital Verfication debloma > Projects > Project2(FIFO WITH UVM) > FIFO_UVM > FIFO_scoreboard_pkg.sv

```
package FIFO_scoreboard_pkg;

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

class FIFO_scoreboard extends uvm_scoreboard;

    `uvm_component_utils(FIFO_scoreboard)

    localparam FIFO_WIDTH = 16;
    localparam FIFO_DEPTH = 8;

    uvm_analysis_export #(FIFO_seq_item) sb_export;
    uvm_tlm_analysis_fifo #(FIFO_seq_item) sb_fifo;
    FIFO_seq_item seq_item_sb;

    logic [FIFO_WIDTH-1:0] fifo_ref[$];
    integer fifo_count = 0;
    logic [FIFO_WIDTH-1:0] data_out_ref;

    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(seq_item_sb);
            reference_model(seq_item_sb);
            #2;
            if (seq_item_sb.data_out !== data_out_ref) begin
                `uvm_error("run_phase", $sformatf(
                    "Comparison failed: DUT output = %0h, Reference output = %0h | Stimulus = %s",
                    seq_item_sb.data_out, data_out_ref, seq_item_sb.convert2string_stimulus()))
                error_count++;
            end else begin
                correct_count++;
                `uvm_info("run_phase", $sformatf("Correct transaction: %s", seq_item_sb.convert2string()), UVM_HIGH);
            end
        end
    endtask
```

```
    // Reference model
    function void reference_model(input FIFO_seq_item F_tx);
        if (!F_tx.rst_n) begin
            fifo_ref <= {};
            fifo_count = 0;

        end else begin
            if (F_tx.wr_en && fifo_count < FIFO_DEPTH) begin
                fifo_ref.push_back(F_tx.data_in);
                fifo_count <= fifo_ref.size();
            end
            if (F_tx.rd_en && fifo_count != 0) begin
                data_out_ref <= fifo_ref.pop_front();
                fifo_count <= fifo_ref.size();
            end

        end
    endfunction

    // 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
```

FIFO_coverage :

D: > Digital Verfication debloma > Projects > Project2(FIFO WITH UVM) > FIFO_UVM > FIFO_coverage_pkg.sv

```
package FIFO_coverage_pkg;
`include "uvm_macros.svh"
import uvm_pkg::*;
import FIFO_sequence_item_pkg::*;
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 cg ;

// Here we create cover point for each signal to be used in the cross coverage
wr_en_cp : coverpoint seq_item_cov.wr_en;
rd_en_cp : coverpoint seq_item_cov.rd_en;
wr_ack_cp : coverpoint seq_item_cov.wr_ack;
full_cp : coverpoint seq_item_cov.full;
empty_cp : coverpoint seq_item_cov.empty;
almostfull_cp : coverpoint seq_item_cov.almostfull;
almostempty_cp : coverpoint seq_item_cov.almostempty;
overflow_cp : coverpoint seq_item_cov.overflow;
underflow_cp : coverpoint seq_item_cov.underflow;

// Here we create cross coverage for each output with read enable & write enable except(dout)
wr_rd_wr_ack_cross : cross wr_en_cp , rd_en_cp , wr_ack_cp
        {
            ignore_bins write_active_with_wr_ack = ! binsof(wr_en_cp) intersect {1} && binsof(wr_ack_cp) intersect {1};
            ignore_bins read_write_active_with_wr_ack = ! binsof(wr_en_cp) intersect {1} && binsof(rd_en_cp) intersect {1} && binsof(wr_ack_cp) intersect {1};
        }

wr_rd_full_cross : cross wr_en_cp , rd_en_cp , full_cp
        {
            ignore_bins write_active_with_full = ! binsof(wr_en_cp) intersect {1} && binsof(full_cp) intersect {1};
            ignore_bins read_active_with_full = binsof(rd_en_cp) intersect {1} && binsof(full_cp) intersect {1};
        }

wr_rd_empty_cross : cross wr_en_cp , rd_en_cp , empty_cp
        {
            ignore_bins read_active_with_empty = ! binsof(rd_en_cp) intersect {1} && binsof(empty_cp) intersect {1};
        }

wr_rd_overflow_cross : cross wr_en_cp , rd_en_cp , overflow_cp
        {
            ignore_bins write_active_with_overflow = ! binsof(wr_en_cp) intersect {1} && binsof(overflow_cp) intersect {1};
        }

wr_rd_underflow_cross : cross wr_en_cp , rd_en_cp , underflow_cp
        {
            ignore_bins read_active_with_underflow = ! binsof(rd_en_cp) intersect {1} && binsof(underflow_cp) intersect {1};
        }

wr_rd_almostfull_cross : cross wr_en_cp , rd_en_cp , almostfull_cp
        {
            ignore_bins write_active_with_almostfull = ! binsof(wr_en_cp) intersect {1} && binsof(almostfull_cp) intersect {1};
            ignore_bins read_write_active_with_almostfull = ! binsof(wr_en_cp) intersect {1} && binsof(rd_en_cp) intersect {1} && binsof(almostfull_cp) intersect {1};
        }

wr_rd_almostempty_cross : cross wr_en_cp , rd_en_cp , almostempty_cp
        {
            ignore_bins read_active_with_almostempty = ! binsof(rd_en_cp) intersect {1} && binsof(almostempty_cp) intersect {1};
            ignore_bins read_write_active_with_almostempty = ! binsof(wr_en_cp) intersect {1} && binsof(rd_en_cp) intersect {1} && binsof(almostempty_cp) intersect {1};
        }

endgroup
```

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


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


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


        task run_phase (uvm_phase phase);
        super.run_phase(phase);
            forever begin
                cov_fifo.get(seq_item_cov);
                cg.sample();
            end
        endtask
endclass
endpackage
```

FIFO_agent:

```
package FIFO_agent_pkg;
  `include "uvm_macros.svh"
  import uvm_pkg::*;
  import FIFO_driver_pkg::*;
  import FIFO_sequencer_pkg::*;
  import FIFO_sequence_item_pkg::*;
  import FIFO_monitor_pkg::*;
  import FIFO_config_obj_pkg::*;
  import FIFO_shared_pkg::*;

  class FIFO_agent extends uvm_agent;
    `uvm_component_utils(FIFO_agent);

    FIFO_object fifo_config_obj_driver;   // ✅ Correct type
    FIFO_driver fifo_dr;
    FIFO_sequencer sqr;
    FIFO_monitor mon;
    uvm_analysis_port #(FIFO_seq_item) agt_ap;

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

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

      if (!uvm_config_db #(FIFO_object)::get(this, "", "CGO", fifo_config_obj_driver)) begin
        `uvm_fatal("build_phase", "DRIVER [] unable to get the virtual interface");
      end


      fifo_dr=FIFO_driver::type_id::create("fifo_dt",this);
      sqr =FIFO_sequencer::type_id::create("sqr",this);
      mon = FIFO_monitor::type_id::create("mon", this);
      agt_ap = new("agt_ap", this);
    endfunction

    function void connect_phase(uvm_phase phase);
      super.connect_phase(phase);
    fifo_dr.seq_item_port.connect(sqr.seq_item_export);
    fifo_dr.sh_vif=fifo_config_obj_driver.fifo_config_vif;
      mon.sh_vif = fifo_config_obj_driver.fifo_config_vif;
      mon.mon_ap.connect(agt_ap);
    endfunction
  endclass
endpackage

```

Fifo_sequence_item:

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

  // Legal: declare parameters at package level
  parameter FIFO_WIDTH = 16;
  parameter FIFO_DEPTH = 8;
  localparam max_fifo_addr = $clog2(FIFO_DEPTH);

  class FIFO_seq_item extends uvm_sequence_item;
    `uvm_object_utils(FIFO_seq_item)

    //_____________ INPUT _____________//
    rand bit [FIFO_WIDTH-1:0] data_in;
    rand bit rst_n, wr_en, rd_en;

    //_____________ OUTPUT _____________//
    bit [FIFO_WIDTH-1:0] data_out;
    bit wr_ack, overflow;
    bit full, empty, almostfull, almostempty, underflow;

    // Control read/write enable bias (sum should not exceed 100)
    int RD_EN_ON_DIST = 30;
    int WR_EN_ON_DIST = 70;

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

    // Constraints
    constraint c_reset {
      rst_n dist {0:/5, 1:/95};
    }

    constraint c_write {
      wr_en dist {0 := (100 - WR_EN_ON_DIST), 1 := WR_EN_ON_DIST};
    }

    constraint c_read {
      rd_en dist {0 := (100 - RD_EN_ON_DIST), 1 := RD_EN_ON_DIST};
    }

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

    function string convert2string_stimulus();
      return $sformatf("%s rst_n=0b%0b, data_in=0b%0b, wr_en=0b%0b, rd_en=0b%0b",
        super.convert2string(), rst_n , data_in , wr_en , rd_en );
    endfunction

  endclass

endpackage

```

Fifo_sequencer:

D: > Digital Verfication debloma > Projects > Project2(FIFO WITH UVM) > FIFO_UVM > FIFO_sequencer_pkg.sv

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

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

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

```

Fifo_sequences:

D: > Digital Verfication debloma > Projects > Project2(FIFO WITH UVM) > FIFO_UVM > FIFO_sequences_pkg.sv

```
package FIFO_sequence_pkg;

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

  //_____________________________________ 1 SEQUENCE _____________________________________//
  class FIFO_reset_sequence extends uvm_sequence#(FIFO_seq_item);
    `uvm_object_utils(FIFO_reset_sequence)
    FIFO_seq_item seq_item;

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

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

  //_____________________________________ 2 SEQUENCE _____________________________________//
  class FIFO_write_only_sequence extends uvm_sequence#(FIFO_seq_item);
    `uvm_object_utils(FIFO_write_only_sequence)
    FIFO_seq_item seq_item;

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

    task body();
      repeat (100) begin
        seq_item = FIFO_seq_item::type_id::create("seq_item");
        start_item(seq_item);
        seq_item.rst_n = 1;
        seq_item.wr_en = 1;
        seq_item.rd_en = 0;
        seq_item.randomize();
        finish_item(seq_item);
      end
    endtask
  endclass
```

```
//_____________________________________ 3 SEQUENCE _____________________________________//
class FIFO_read_only_sequence extends uvm_sequence#(FIFO_seq_item);
  `uvm_object_utils(FIFO_read_only_sequence)
  FIFO_seq_item seq_item;

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

  task body();
    repeat (100) begin
      seq_item = FIFO_seq_item::type_id::create("seq_item");
      start_item(seq_item);
      seq_item.rst_n = 1;
      seq_item.wr_en = 0;
      seq_item.rd_en = 1;
      seq_item.randomize();
      finish_item(seq_item);
    end
  endtask
endclass

//_____________________________________ 4 SEQUENCE _____________________________________//
class FIFO_read_write_sequence extends uvm_sequence#(FIFO_seq_item);
  `uvm_object_utils(FIFO_read_write_sequence)
  FIFO_seq_item seq_item;

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

  task body();
    repeat (100) begin
      seq_item = FIFO_seq_item::type_id::create("seq_item");
      start_item(seq_item);
      seq_item.randomize();
      finish_item(seq_item);
    end
  endtask
endclass
```

```
90    //_____________________________________ S SEQUENCE _____________________________________//
91    class FIFO_read_write_empty_sequence extends uvm_sequence#(FIFO_seq_item);
92      `uvm_object_utils(FIFO_read_write_empty_sequence)
93      FIFO_seq_item seq_item;
94
95      function new(string name = "FIFO_read_write_empty_sequence");
96        super.new(name);
97      endfunction
98
99      task body();
00
01        // Write until FIFO is full
02        for (int i = 0; i < FIFO_DEPTH; i++) begin
03          seq_item = FIFO_seq_item::type_id::create("seq_item");
04          start_item(seq_item);
05          seq_item.rst_n = 1;
06          seq_item.wr_en = 1;
07          seq_item.rd_en = 0;
08          seq_item.randomize();
09          finish_item(seq_item);
10        end
11
12        // Read until FIFO is empty
13        for (int i = 0; i < FIFO_DEPTH; i++) begin
14          seq_item = FIFO_seq_item::type_id::create("seq_item");
15          start_item(seq_item);
16          seq_item.rst_n = 1;
17          seq_item.wr_en = 0;
18          seq_item.rd_en = 1;
19          seq_item.randomize();
20          finish_item(seq_item);
21        end
22
23        // Final simultaneous read/write
24        seq_item = FIFO_seq_item::type_id::create("seq_item");
25        start_item(seq_item);
26        seq_item.rst_n = 1;
27        seq_item.wr_en = 1;
28        seq_item.rd_en = 1;
29        seq_item.randomize();
30        finish_item(seq_item);
31      endtask
32    endclass
33
34  endpackage
35
```

FIFO_driver:

```
package FIFO_driver_pkg;
  `include "uvm_macros.svh"
  import uvm_pkg::*;
  import FIFO_sequence_item_pkg::*;
  import FIFO_config_obj_pkg::*;

  class FIFO_driver extends uvm_driver#(FIFO_seq_item);
    `uvm_component_utils(FIFO_driver)

    FIFO_seq_item stim_seq_item;
    virtual FIFO_if sh_vif;

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

    task run_phase(uvm_phase phase);
      super.run_phase(phase);
      forever begin
        // Create a new sequence item
        stim_seq_item = FIFO_seq_item::type_id::create("stim_seq_item");

        // Get transaction from sequencer
        seq_item_port.get_next_item(stim_seq_item);

        // Drive stimulus to DUT interface
        sh_vif.rst_n        = stim_seq_item.rst_n;
        sh_vif.wr_en        = stim_seq_item.wr_en;
        sh_vif.rd_en        = stim_seq_item.rd_en;
        sh_vif.data_in      = stim_seq_item.data_in;


        // Wait for one negative clock edge
        @(negedge sh_vif.clk);

        // Capture DUT output
        stim_seq_item.wr_ack  = sh_vif.wr_ack;
        stim_seq_item.overflow = sh_vif.overflow;
         stim_seq_item.underflow = sh_vif.underflow;
          stim_seq_item.full = sh_vif.full;
           stim_seq_item.empty = sh_vif.empty;
            stim_seq_item.almostfull = sh_vif.almostfull;
             stim_seq_item.almostempty = sh_vif.almostempty;
              stim_seq_item.data_out = sh_vif.data_out;

        // Complete transaction
        seq_item_port.item_done();

        `uvm_info("run_phase", stim_seq_item.convert2string_stimulus(), UVM_HIGH)
      end
    endtask
  endclass
endpackage
```

FIFO_monitor:

D: > Digital Verfication debloma > Projects > Project2(FIFO WITH UVM) > FIFO_UVM > FIFO_monitor_pkg.sv

```
package FIFO_monitor_pkg;

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

class FIFO_monitor extends uvm_monitor;
    `uvm_component_utils(FIFO_monitor);

    virtual FIFO_if sh_vif;
    FIFO_seq_item rsp_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 // new()

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

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever begin
            rsp_seq_item = FIFO_seq_item::type_id::create("rsp_seq_item");

            // Ensure you have correct field names here that match those in the FIFO_SEQ_ITEM class
            @(negedge sh_vif.clk);

            rsp_seq_item.rst_n = sh_vif.rst_n;
            rsp_seq_item.wr_en = sh_vif.wr_en;
            rsp_seq_item.rd_en = sh_vif.rd_en;
            rsp_seq_item.data_in = sh_vif.data_in;
            rsp_seq_item.data_out = sh_vif.data_out;
            rsp_seq_item.wr_ack = sh_vif.wr_ack;
            rsp_seq_item.overflow = sh_vif.overflow;
            rsp_seq_item.underflow = sh_vif.underflow;
            rsp_seq_item.full = sh_vif.full;
            rsp_seq_item.empty= sh_vif.empty;
            rsp_seq_item.almostfull = sh_vif.almostfull;
            rsp_seq_item.almostempty = sh_vif.almostempty;

            // Write the sequence item to the analysis port
            mon_ap.write(rsp_seq_item);

            // Log the converted string
            `uvm_info("run_phase", rsp_seq_item.convert2string(), UVM_HIGH);
        end
    endtask
endclass

endpackage
```

FIFO_configuration_object:

D: > Digital Verfication debloma > Projects > Project2(FIFO WITH UVM) > FIFO_UVM > FIFO_config_obj_pkg.sv

```
1   package FIFO_config_obj_pkg;
2   `include "uvm_macros.svh"
3   import uvm_pkg::*; // Corrected the package name from unm_pkg to uvm_pkg
4
5   class FIFO_object extends uvm_object;
6       `uvm_object_utils(FIFO_object);
7
8       virtual FIFO_if fifo_config_vif;
9       uvm_active_passive_enum sel_mod;
10
11      function new(string name = "FIFO_object");
12          super.new(name);
13      endfunction //new()
14  endclass //FIFO_object
15  endpackage
16
```

Shared _pkg:

D: > Digital Verfication debloma > Projects > Project2(FIFO WIT

```
1   package FIFO_shared_pkg;
2   integer error_count=0;
3   integer correct_count=0;
4
5   endpackage
```

Do file :

D: > Digital Verfication debloma > Projects > Project2(FIFO WITH UVM) > FIFO_UVM > run.do

```
1   vlib work
2   vlog *v +cover
3   vsim -voptargs=+acc work.FIFO_top -classdebug -uvmcontrol=all -cover
4   add wave /FIFO_top/clk
5   add wave /FIFO_top/if_obj/*
6   coverage save FIFO_top.ucdb *onexit
7   run -all
8   quit -sim
9   vcover report FIFO.ucdb -all -annotate -details  -output coverage.txt
```

List of files:

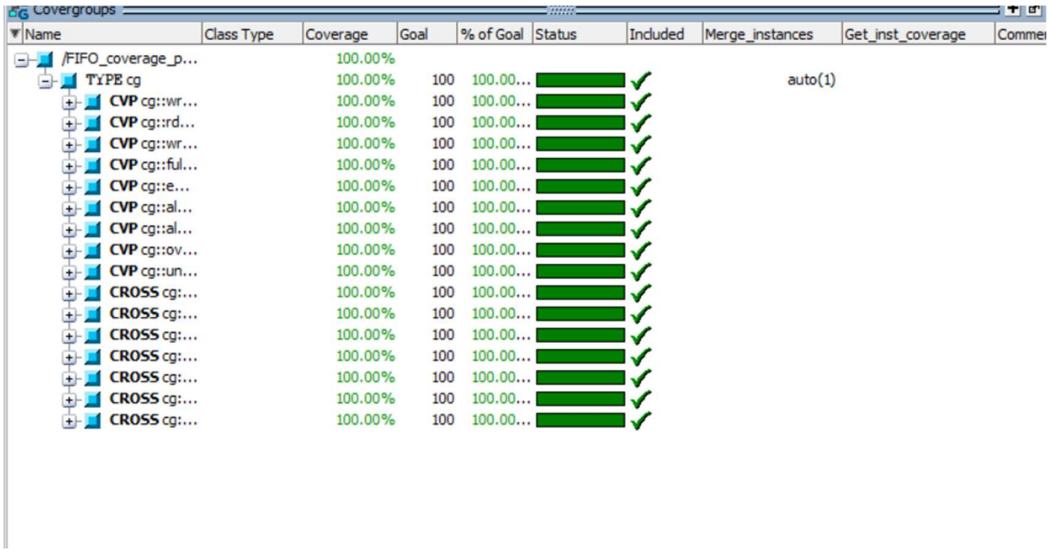
```
FIFO_after.sv
FIFO_SVA.sv
FIFO_interface.sv
FIFO_shared_pkg.sv
FIFO_config_pkg.sv
FIFO_sequence_item_pkg.sv
FIFO_sequencer_pkg.sv
FIFO_sequences_pkg.sv
FIFO_driver_pkg.sv
FIFO_monitor_pkg.sv
FIFO_agent_pkg.sv
FIFO_scoreboard_pkg.sv
FIFO_coverage_pkg.sv
FIFO_env_pkg.sv
FIFO_test_pkg.sv
FIFO_top.sv
```

Assertion :

Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Count	Memory	Peak Memory	Peak Memory Time	Cumulative Threads	ATV	Assertion Expression	Included
/uvm_pkg::uvm_re...	Immediate	SVA	on	0	0	-	-	-	-		off	assert ($cast(seq,o))	✗
/uvm_pkg::uvm_re...	Immediate	SVA	on	0	0	-	-	-	-		off	assert ()	✗
/FIFO_top/DUT/SA...	Immediate	SVA	on	0	1	-	-	-	-		off	assert ((if_obj.wr_ack~|if_obj.over...	✓
/FIFO_top/DUT/SA...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (if_obj.full)	✓
/FIFO_top/DUT/SA...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (if_obj.empty)	✓
/FIFO_top/DUT/SA...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (if_obj.almostfull)	✓
/FIFO_top/DUT/SA...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (if_obj.almostempty)	✓
/FIFO_top/DUT/SA...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge if_obj.clk,neged...	✓
/FIFO_top/DUT/SA...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge if_obj.clk) disa...	✓
/FIFO_top/DUT/SA...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge if_obj.clk) disa...	✓
/FIFO_top/DUT/SA...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge if_obj.clk) disa...	✓
/FIFO_top/DUT/SA...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge if_obj.clk) disa...	✓
/FIFO_top/DUT/SA...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge if_obj.clk) disa...	✓
/FIFO_top/DUT/SA...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge if_obj.clk) disa...	✓
/FIFO_top/DUT/SA...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge if_obj.clk) disa...	✓
/FIFO_top/DUT/SA...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge if_obj.clk) disa...	✓

Note :the first two assertion (ignor)

Function coverage :




```
Coverage Report by instance with details

================================================================================
=== Instance: /FIFO_top/DUT/SAV
=== Design Unit: work.FIFO_SVA
================================================================================

Directive Coverage:
    Directives                      14        14         0   100.00%

DIRECTIVE COVERAGE:
--------------------------------------------------------------------------------------------
Name                                 Design Design   Lang File(Line)      Hits Status
                                     Unit   UnitType
--------------------------------------------------------------------------------------------
/FIFO_top/DUT/SAV/c1                 FIFO_SVA Verilog  SVA  FIFO_SVA.sv(12)   13 Covered
/FIFO_top/DUT/SAV/FULL_COVER         FIFO_SVA Verilog  SVA  FIFO_SVA.sv(21)   39 Covered
/FIFO_top/DUT/SAV/empty_COVER        FIFO_SVA Verilog  SVA  FIFO_SVA.sv(29)   16 Covered
/FIFO_top/DUT/SAV/almostfull_COVER   FIFO_SVA Verilog  SVA  FIFO_SVA.sv(37)   46 Covered
/FIFO_top/DUT/SAV/almostempty_COVER  FIFO_SVA Verilog  SVA  FIFO_SVA.sv(45)   16 Covered
/FIFO_top/DUT/SAV/c2                 FIFO_SVA Verilog  SVA  FIFO_SVA.sv(56)   13 Covered
/FIFO_top/DUT/SAV/c3                 FIFO_SVA Verilog  SVA  FIFO_SVA.sv(66)  152 Covered
/FIFO_top/DUT/SAV/c4                 FIFO_SVA Verilog  SVA  FIFO_SVA.sv(74)   62 Covered
/FIFO_top/DUT/SAV/c5                 FIFO_SVA Verilog  SVA  FIFO_SVA.sv(84)   77 Covered
/FIFO_top/DUT/SAV/c6                 FIFO_SVA Verilog  SVA  FIFO_SVA.sv(92)    8 Covered
/FIFO_top/DUT/SAV/c7                 FIFO_SVA Verilog  SVA  FIFO_SVA.sv(101) 111 Covered
/FIFO_top/DUT/SAV/c8                 FIFO_SVA Verilog  SVA  FIFO_SVA.sv(110)  17 Covered
/FIFO_top/DUT/SAV/c9                 FIFO_SVA Verilog  SVA  FIFO_SVA.sv(119)  25 Covered
/FIFO_top/DUT/SAV/c10                FIFO_SVA Verilog  SVA  FIFO_SVA.sv(128)   6 Covered

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

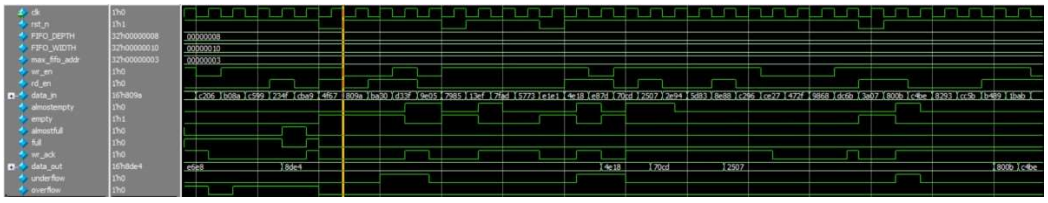
```
================================================================================

Covergroup Coverage:
    Covergroups                     1        na        na   100.00%
        Coverpoints/Crosses        16        na        na        na
            Covergroup Bins        58        58         0   100.00%
----------------------------------------------------------------------------------------------------------------
Covergroup                                          Metric       Goal       Bins    Status

----------------------------------------------------------------------------------------------------------------
 TYPE /FIFO_coverage_pkg/FIFO_coverage/cg          100.00%        100          -    Covered
    covered/total bins:                                 58         58          -
    missing/total bins:                                  0         58          -
    % Hit:                                         100.00%        100          -
    Coverpoint wr_en_cp                            100.00%        100          -    Covered
        covered/total bins:                              2          2          -
        missing/total bins:                              0          2          -
        % Hit:                                     100.00%        100          -
        bin auto[0]                                     88          1          -    Covered
        bin auto[1]                                    230          1          -    Covered
    Coverpoint rd_en_cp                            100.00%        100          -    Covered
        covered/total bins:                              2          2          -
        missing/total bins:                              0          2          -
        % Hit:                                     100.00%        100          -
        bin auto[0]                                    227          1          -    Covered
        bin auto[1]                                     91          1          -    Covered
    Coverpoint wr_ack_cp                           100.00%        100          -    Covered
        covered/total bins:                              2          2          -
        missing/total bins:                              0          2          -
        % Hit:                                     100.00%        100          -
        bin auto[0]                                    161          1          -    Covered
        bin auto[1]                                    157          1          -    Covered
    Coverpoint full_cp                             100.00%        100          -    Covered
        covered/total bins:                              2          2          -
        missing/total bins:                              0          2          -
        % Hit:                                     100.00%        100          -
        bin auto[0]                                    220          1          -    Covered
        bin auto[1]                                     98          1          -    Covered
    Coverpoint empty_cp                            100.00%        100          -    Covered
        covered/total bins:                              2          2          -
            ignore_bin read_active_with_almostempty
                                                        12                     -    Occurred

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

----------------------------------------------------------------------------------------------------------------
 TYPE /FIFO_coverage_pkg/FIFO_coverage/cg          100.00%        100          -    Covered
    covered/total bins:                                 58         58          -
    missing/total bins:                                  0         58          -
    % Hit:                                         100.00%        100          -
    Coverpoint wr_en_cp                            100.00%        100          -    Covered
        covered/total bins:                              2          2          -
        missing/total bins:                              0          2          -
        % Hit:                                     100.00%        100          -
        bin auto[0]                                     88          1          -    Covered
        bin auto[1]                                    230          1          -    Covered
    Coverpoint rd_en_cp                            100.00%        100          -    Covered
        covered/total bins:                              2          2          -
        missing/total bins:                              0          2          -
        % Hit:                                     100.00%        100          -
        bin auto[0]                                    227          1          -    Covered
        bin auto[1]                                     91          1          -    Covered
    Coverpoint wr_ack_cp                           100.00%        100          -    Covered
        covered/total bins:                              2          2          -
        missing/total bins:                              0          2          -
        % Hit:                                     100.00%        100          -
        bin auto[0]                                    161          1          -    Covered
        bin auto[1]                                    157          1          -    Covered
    Coverpoint full_cp                             100.00%        100          -    Covered
        covered/total bins:                              2          2          -
        missing/total bins:                              0          2          -
        % Hit:                                     100.00%        100          -
        bin auto[0]                                    220          1          -    Covered
        bin auto[1]                                     98          1          -    Covered
    Coverpoint empty_cp                            100.00%        100          -    Covered
        covered/total bins:                              2          2          -
        missing/total bins:                              0          2          -
        % Hit:                                     100.00%        100          -
```

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

DIRECTIVE COVERAGE:
--------------------------------------------------------------------------------------------
Name                                    Design Design   Lang File(Line)      Hits Status
                                        Unit   UnitType
--------------------------------------------------------------------------------------------
/FIFO_top/DUT/SAV/c1                    FIFO_SVA Verilog  SVA  FIFO_SVA.sv(12)    13 Covered
/FIFO_top/DUT/SAV/FULL_COVER            FIFO_SVA Verilog  SVA  FIFO_SVA.sv(21)    39 Covered
/FIFO_top/DUT/SAV/empty_COVER           FIFO_SVA Verilog  SVA  FIFO_SVA.sv(29)    16 Covered
/FIFO_top/DUT/SAV/almostfull_COVER      FIFO_SVA Verilog  SVA  FIFO_SVA.sv(37)    46 Covered
/FIFO_top/DUT/SAV/almostempty_COVER     FIFO_SVA Verilog  SVA  FIFO_SVA.sv(45)    16 Covered
/FIFO_top/DUT/SAV/c2                    FIFO_SVA Verilog  SVA  FIFO_SVA.sv(56)    13 Covered
/FIFO_top/DUT/SAV/c3                    FIFO_SVA Verilog  SVA  FIFO_SVA.sv(66)   152 Covered
/FIFO_top/DUT/SAV/c4                    FIFO_SVA Verilog  SVA  FIFO_SVA.sv(74)    62 Covered
/FIFO_top/DUT/SAV/c5                    FIFO_SVA Verilog  SVA  FIFO_SVA.sv(84)    77 Covered
/FIFO_top/DUT/SAV/c6                    FIFO_SVA Verilog  SVA  FIFO_SVA.sv(92)     8 Covered
/FIFO_top/DUT/SAV/c7                    FIFO_SVA Verilog  SVA  FIFO_SVA.sv(101) 111 Covered
/FIFO_top/DUT/SAV/c8                    FIFO_SVA Verilog  SVA  FIFO_SVA.sv(110)  17 Covered
/FIFO_top/DUT/SAV/c9                    FIFO_SVA Verilog  SVA  FIFO_SVA.sv(119)  25 Covered
/FIFO_top/DUT/SAV/c10                   FIFO_SVA Verilog  SVA  FIFO_SVA.sv(128)   6 Covered

TOTAL DIRECTIVE COVERAGE: 100.00%  COVERS: 14

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

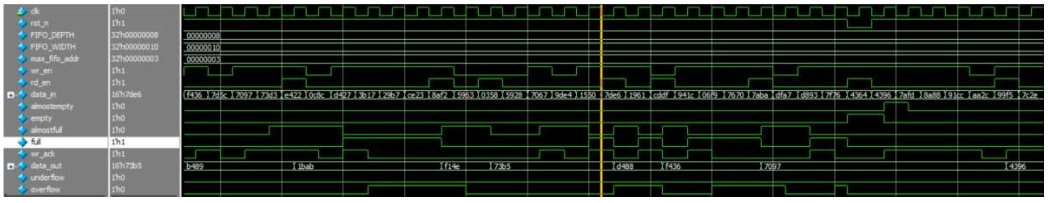
QuestaSim simulation waveform & transcript snippets:

Labels snippets:

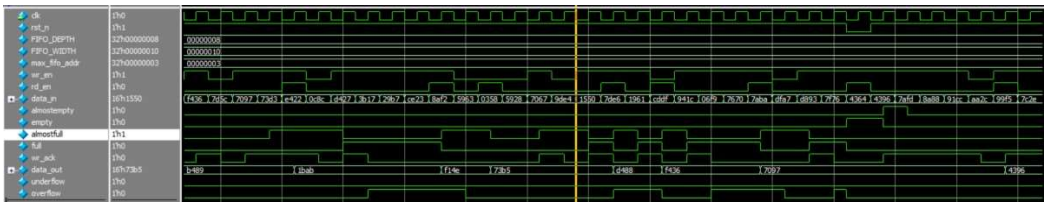
Label1(When the rst_n is asserted, All flags & internal signals should equal 0):



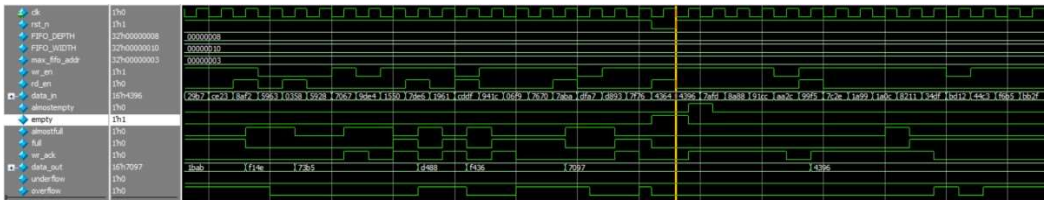
Label2(When rst_n is deactivated & the FIFO is full of elements (conut=depth), The full flag should be high):



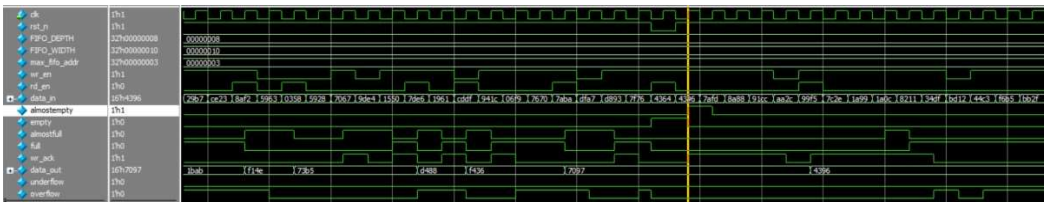
Label3(When rst_n is deactivated & the FIFO has one element left free(conut=depth-1), The almostfull flag should be high):



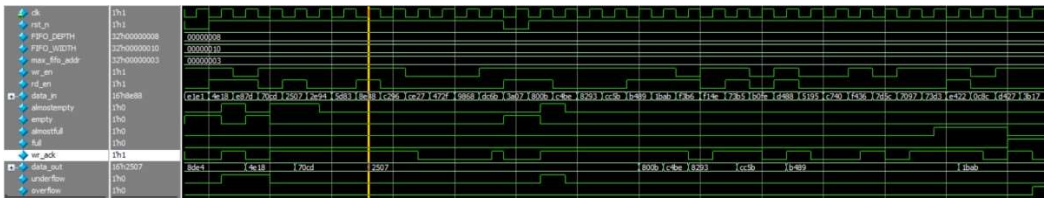
Label4(When rst_n is deactivated & the FIFO has no element inside (conut=0), The empty flag should be high):



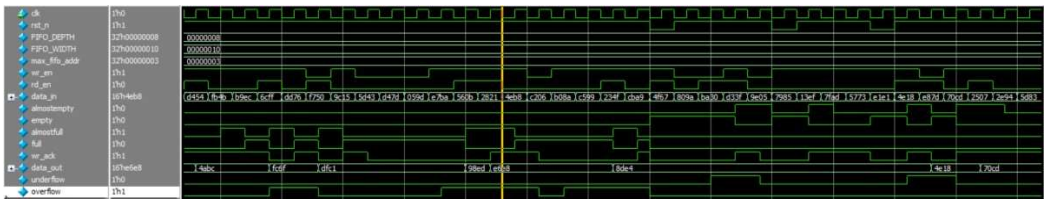
Label5(When rst_n is deactivated & the FIFO has only one element inside (conut=1), The empty flag should be high):



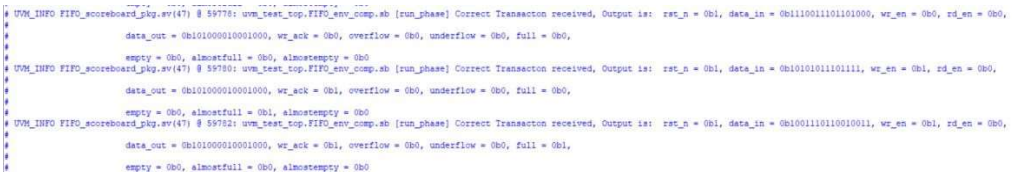
Label6(When rst_n is deactivated, You want to read & the FIFO is not full, The wr_ack flag should be high and wr_ptr should increment):



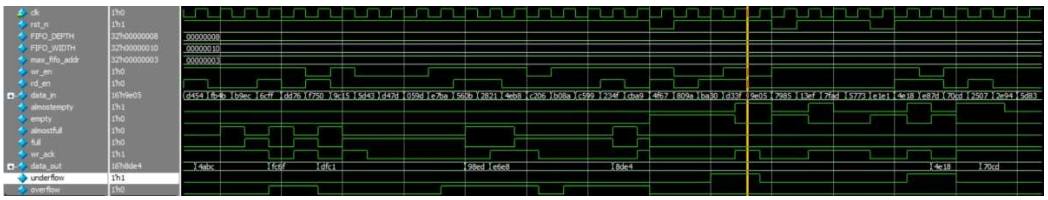
Label7(When rst_n is deactivated, You want to read & the FIFO is full, The overflow flag should be high):



Label8(When rst_n is deactivated, You want to read & the FIFO is not empty rd_ptr should increment and data_out should equal mem[rd_ptr]):



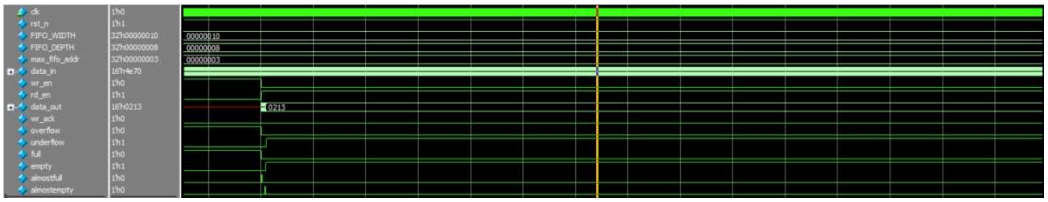
Label9(When rst_n is deactivated, You want to read & the FIFO is empty underflow flag should be high):



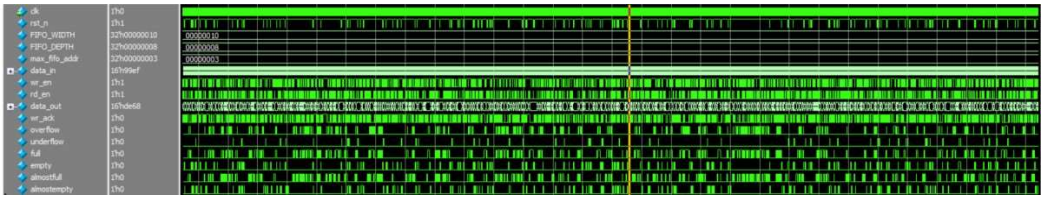
Sequences snippets:

Reset & Write only sequences:

Read only sequence:



Write & Read together sequences:



transcript snippets:

```
# UVM_ERROR D:/Digital Verfication debloma/Projects/Project2(FIFO WITH UVM)/FIFO_UVM/FIFO_scoreboard_pkg.sv(52) @ 8: uvm_test_top.env.sb [run_pha
 =  rst_n=0b1, data_in=0b1010000110010010, wr_en=0b1, rd_en=0b0
# UVM_INFO D:/Digital Verfication debloma/Projects/Project2(FIFO WITH UVM)/FIFO_UVM/FIFO_test_pkg.sv(60) @ 202: uvm_test_top [run_phase] WRITE er
# UVM_INFO D:/Digital Verfication debloma/Projects/Project2(FIFO WITH UVM)/FIFO_UVM/FIFO_test_pkg.sv(63) @ 202: uvm_test_top [run_phase] READ sta
# UVM_INFO D:/Digital Verfication debloma/Projects/Project2(FIFO WITH UVM)/FIFO_UVM/FIFO_test_pkg.sv(65) @ 402: uvm_test_top [run_phase] READ enc
# UVM_INFO D:/Digital Verfication debloma/Projects/Project2(FIFO WITH UVM)/FIFO_UVM/FIFO_test_pkg.sv(69) @ 402: uvm_test_top [run_phase] READ WRI
# UVM_INFO D:/Digital Verfication debloma/Projects/Project2(FIFO WITH UVM)/FIFO_UVM/FIFO_test_pkg.sv(71) @ 602: uvm_test_top [run_phase] READ WRI
# UVM_INFO D:/Digital Verfication debloma/Projects/Project2(FIFO WITH UVM)/FIFO_UVM/FIFO_test_pkg.sv(75) @ 602: uvm_test_top [run_phase] READ WRI
# UVM_INFO D:/Digital Verfication debloma/Projects/Project2(FIFO WITH UVM)/FIFO_UVM/FIFO_test_pkg.sv(77) @ 636: uvm_test_top [run_phase] READ WRI
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 636: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' pha
# UVM_INFO D:/Digital Verfication debloma/Projects/Project2(FIFO WITH UVM)/FIFO_UVM/FIFO_scoreboard_pkg.sv(85) @ 636: uvm_test_top.env.sb [report
# UVM_INFO D:/Digital Verfication debloma/Projects/Project2(FIFO WITH UVM)/FIFO_UVM/FIFO_scoreboard_pkg.sv(86) @ 636: uvm_test_top.env.sb [report
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   16
# UVM_WARNING :    0
# UVM_ERROR :    3
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# [report_phase]     2
# [run_phase]    13
# ** Note: $finish    : C:/questasim64_2024.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 636 ns  Iteration: 61  Instance: /FIFO_top
# 1
# Break in Task uvm_pkg/uvm_root::run_test at C:/questasim64_2024.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh line 430

VSIM 10>
```

```
#      (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 D:/Digital Verfication debloma/Projects/Project2(FIFO WITH UVM)/FIFO_UVM/FIFO_test_pkg.sv(53) @ 0: uvm_test_top [run_phase] reset asserted
# UVM_INFO D:/Digital Verfication debloma/Projects/Project2(FIFO WITH UVM)/FIFO_UVM/FIFO_test_pkg.sv(55) @ 2: uvm_test_top [run_phase] reset deasserted
# UVM_INFO D:/Digital Verfication debloma/Projects/Project2(FIFO WITH UVM)/FIFO_UVM/FIFO_test_pkg.sv(58) @ 2: uvm_test_top [run_phase] WRITE started
# UVM_ERROR D:/Digital Verfication debloma/Projects/Project2(FIFO WITH UVM)/FIFO_UVM/FIFO_scoreboard_pkg.sv(52) @ 4: uvm_test_top.env.sb [run_phase] Comparison failed: DUT output = 0, Reference output = x | Stimulu
 =  rst_n=0b0, data_in=0b0, wr_en=0b0, rd_en=0b0
# UVM_ERROR D:/Digital Verfication debloma/Projects/Project2(FIFO WITH UVM)/FIFO_UVM/FIFO_scoreboard_pkg.sv(52) @ 6: uvm_test_top.env.sb [run_phase] Comparison failed: DUT output = 0, Reference output = x | Stimulu
 =  rst_n=0b1, data_in=0b1001001101101111, wr_en=0b0, rd_en=0b0
# UVM_ERROR D:/Digital Verfication debloma/Projects/Project2(FIFO WITH UVM)/FIFO_UVM/FIFO_scoreboard_pkg.sv(52) @ 8: uvm_test_top.env.sb [run_phase] Comparison failed: DUT output = 0, Reference output = x | Stimulu
 =  rst_n=0b1, data_in=0b1010000110010010, wr_en=0b1, rd_en=0b0
# UVM_INFO D:/Digital Verfication debloma/Projects/Project2(FIFO WITH UVM)/FIFO_UVM/FIFO_test_pkg.sv(60) @ 202: uvm_test_top [run_phase] WRITE ended
# UVM_INFO D:/Digital Verfication debloma/Projects/Project2(FIFO WITH UVM)/FIFO_UVM/FIFO_test_pkg.sv(63) @ 202: uvm_test_top [run_phase] READ started
# UVM_INFO D:/Digital Verfication debloma/Projects/Project2(FIFO WITH UVM)/FIFO_UVM/FIFO_test_pkg.sv(65) @ 402: uvm_test_top [run_phase] READ ended
# UVM_INFO D:/Digital Verfication debloma/Projects/Project2(FIFO WITH UVM)/FIFO_UVM/FIFO_test_pkg.sv(69) @ 402: uvm_test_top [run_phase] READ WRITE started
# UVM_INFO D:/Digital Verfication debloma/Projects/Project2(FIFO WITH UVM)/FIFO_UVM/FIFO_test_pkg.sv(71) @ 602: uvm_test_top [run_phase] READ WRITE ended
# UVM_INFO D:/Digital Verfication debloma/Projects/Project2(FIFO WITH UVM)/FIFO_UVM/FIFO_test_pkg.sv(75) @ 602: uvm_test_top [run_phase] READ WRITE EMPTY started
# UVM_INFO D:/Digital Verfication debloma/Projects/Project2(FIFO WITH UVM)/FIFO_UVM/FIFO_test_pkg.sv(77) @ 636: uvm_test_top [run_phase] READ WRITE EMPTY ended
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 636: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO D:/Digital Verfication debloma/Projects/Project2(FIFO WITH UVM)/FIFO_UVM/FIFO_scoreboard_pkg.sv(85) @ 636: uvm_test_top.env.sb [report_phase] Total successful transactions: 314
# UVM_INFO D:/Digital Verfication debloma/Projects/Project2(FIFO WITH UVM)/FIFO_UVM/FIFO_scoreboard_pkg.sv(86) @ 636: uvm_test_top.env.sb [report_phase] Total failed transactions: 3
#
```

Coverage directives:

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/SA...	SVA	✓	Off	13	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/SA...	SVA	✓	Off	39	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/SA...	SVA	✓	Off	16	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/SA...	SVA	✓	Off	46	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/SA...	SVA	✓	Off	16	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/SA...	SVA	✓	Off	13	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/SA...	SVA	✓	Off	152	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/SA...	SVA	✓	Off	62	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/SA...	SVA	✓	Off	77	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/SA...	SVA	✓	Off	8	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/SA...	SVA	✓	Off	111	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/SA...	SVA	✓	Off	17	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/SA...	SVA	✓	Off	25	1	Unli...	1	100%		✓	0	0	0 ns	0
/FIFO_top/DUT/SA...	SVA	✓	Off	6	1	Unli...	1	100%		✓	0	0	0 ns	0

Analysis:

Code Coverage Analysis

Statements - by instance (/FIFO_top)

FIFO_top.sv

```
 7 clk=0;
 8 forever begin
 9 #1 clk = ~clk;
 9.1 #
 9.2 clk
18 uvm_config_db#(virtual FIFO_if)::set(null,"uvm_test_top","FIFO_IF",if_obj);
19 run_test("FIFO_test");
```

تم بحمد الله

(وَقُلِ ٱعْمَلُواْ فَسَيَرَى ٱللَّهُ عَمَلَكُمْ وَرَسُولُهُۥ وَٱلْمُؤْمِنُونَ)

سورة التوبة، الآية ١٠٥

صدق الله العظيم