Synchronous FIFO

Digital Verification Using SV and UVM

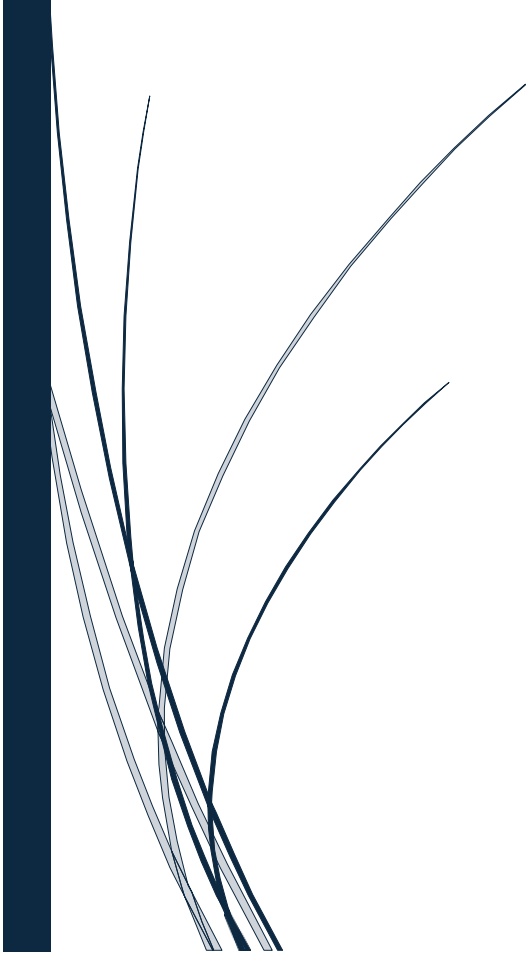


Table of Contents: -

Introduction • Definition of FIFO • Functionality of FIFO	Page 3,4,5
Applications of FIFO	Page 6
Verification Plan and Diagram	Page 7
HDL (System Verilog) Verification Content • FIFO interface • FIFO design module • FIFO verification modules • FIFO verification packages • FIFO do file	Page (8□21)
Code Coverage • Branch coverage • Statement coverage • Toggle coverage	Page 22,23
Functional Coverage • Cross Coverage	Page 24
Questa Sim Snippets • Reading Test case • Writing Test case • Snippet to all iterations • Assertions • Cover directives • Transcript Error count and Correct count	Page 25,26,27,28

Introduction:

FIFO Definition: A FIFO (First-In, First-Out) is a type of data structure or queue used in digital systems, memory management, and communication protocols. It operates on the principle that the first data element to be inserted is the first one to be retrieved, like how a queue in real life works.

FIFO Functionality:

1. **Basic Operations in FIFO:**

A FIFO structure typically supports two main operations:

- **Enqueue (Write):** This operation adds data to the FIFO. The new data is placed at the **end (tail)** of the queue.
- **Dequeue (Read):** This operation removes data from the FIFO. The oldest data, i.e., the data at the **front (head)** of the queue, is the one to be retrieved and removed.

2. **Pointers (Head and Tail):**

- **Head Pointer:** Tracks the position of the next data to be read or dequeued.
- **Tail Pointer:** Tracks the position where new data will be written or enqueued.

The FIFO functionality uses these pointers to manage data flow efficiently.

3. **Empty and Full Conditions:**

- **Empty Condition:** The FIFO is considered empty when there is no data to be read, i.e., the **head pointer** equals the **tail pointer** (there's no data in the buffer).
- **Full Condition:** The FIFO is full when no additional data can be written, i.e., the **tail pointer** has wrapped around and caught up to the **head pointer**, depending on the specific design (circular buffer concept).

4. **Clock and Timing (for Synchronous FIFO):**

In synchronous FIFOs, both reading (dequeue) and writing (enqueue) are governed by the same clock. The operations are typically triggered on the clock edge (rising or falling edge). Timing plays a crucial role in determining when data can be written or read from the FIFO.

5. **FIFO Buffer Size:**

The **size of the FIFO** determines how much data it can hold. When a FIFO is full, any attempt to add more data will either be blocked (in hardware FIFO) or may overwrite old data (depending on how the system is configured). In many designs, a status flag (e.g., "full" or "empty") is used to indicate whether the FIFO can accept new data or has data ready for reading.

6. **Data Flow Control:**

- **Flow Control**: In communication systems, a FIFO can help regulate data flow between devices operating at different speeds (e.g., between a faster producer and a slower consumer).
- **Backpressure**: When the FIFO is full, it can signal the producer to stop or slow down data production.

7. FIFO in a Circular Buffer:

In many designs, FIFO operates as a **circular buffer**, where the tail pointer wraps around to the start of the buffer when it reaches the end. This allows for efficient memory utilization without shifting data.

8. FIFO Flags and Status Indicators:

- **Empty Flag**: Indicates that no data is available to be read.
- **Full Flag**: Indicates that no more data can be written until some is read out.
- **Almost Full/Almost Empty Flags**: Some FIFOs have these to give early warning before the full or empty conditions occur, allowing for better flow control.

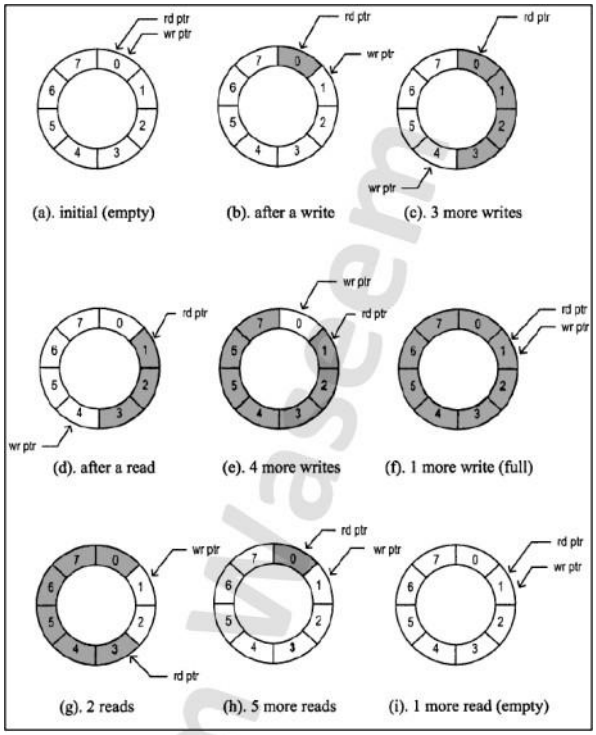


Figure 1 FIFO With depth of 8 words

Applications of FIFO: -

1. **Clock Domain Crossing (CDC):**

   FIFOs are essential in systems where different clock domains exist, such as in multi-clock FPGAs or SoCs. They provide a way to transfer data across these domains safely without risking data corruption.

2. **Data Buffering in Communication Protocols:**

   FIFO buffers are used extensively in communication protocols like UART, SPI, and Ethernet, where they help smooth data transmission and reception across devices.

3. **Audio/Video Streaming:**

   In audio or video streaming, where real-time data flow is critical, FIFOs are used to handle continuous data streams, ensuring that data is processed sequentially, and no frames are skipped.

4. **Processors and Microcontrollers:**

   Processors use FIFOs in their instruction pipelines and memory management units to handle instructions and data efficiently, allowing for smooth multitasking and efficient use of system resources.

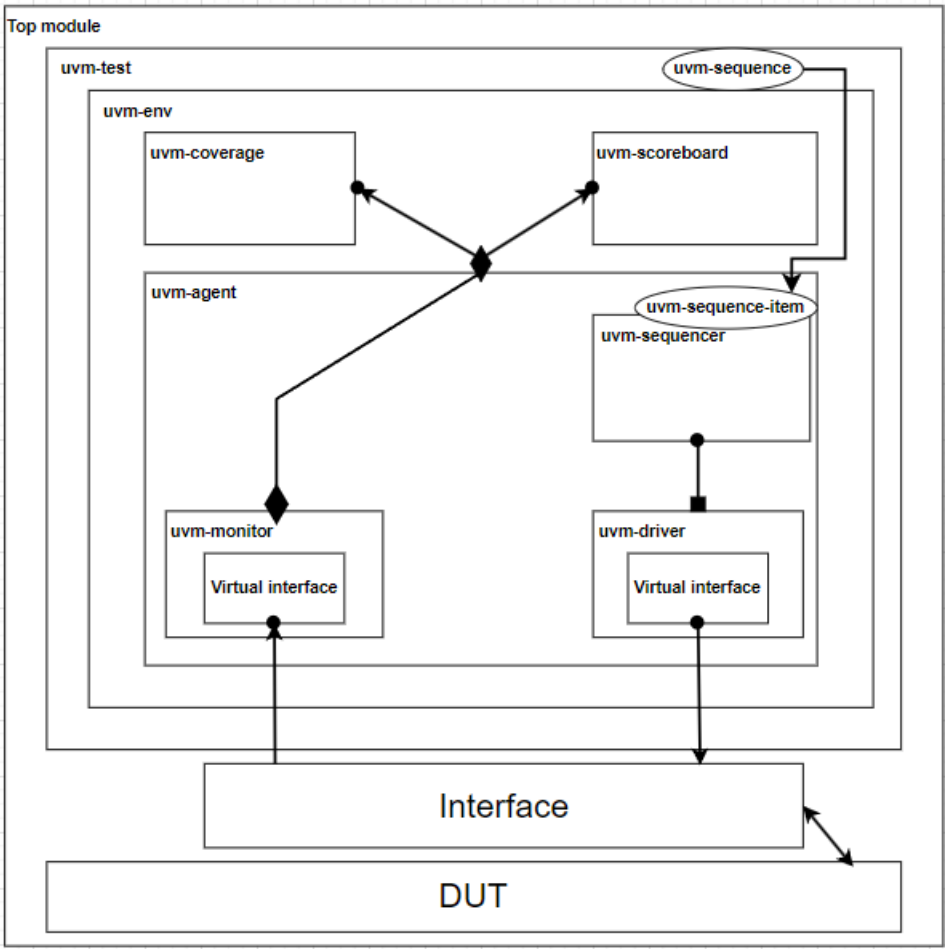
5. **Networking:**

   In routers and switches, FIFOs help manage packet flows, ensuring that network packets are transmitted in the correct order and preventing
   packet loss in congested networks. 6. Image Processing: FIFOs are useful in image processing applications, where large sets of pixel data need to be buffered and processed sequentially to ensure accurate
   rendering.

Verification Plan: -

Label	Description	Stimulus Generation	Functional Coverage	Functionality Check
FIFO_1	Testing the Active low Async reset, when it's active outputs must be low while constraining reset signal to be inactive most of the time	Directed testing at the start of the simulation		Assertion in the design to check that the outputs are low
FIFO_2	Testing that Overflow flag is high if write enable is high and FIFO is full	Randomized testing throughout the simulation	bins for cross coverage between wr_en and overflow	Assertion in the design to check that Overflow is high
FIFO_3	Testing that Underflow flag is high if read enable is high and FIFO is empty	Randomized testing throughout the simulation	bins for cross coverage between rd_en and underflow	Assertion in the design to check that Underflow is high
FIFO_4	Testing that write acknowledge is high if write operation is done (Write_enable is high and FIFO is not full)	Randomized testing throughout the simulation	bins for cross coverage between wr_en and wr_ack	Assertion in the design to check that Wr_ack is high
FIFO_5	Testing that full flag is high when count equal to FIFO_DEPTH	Randomized testing throughout the simulation	bins for cross coverage between wr_en, rd_en and Full	Assertion in the design to check that full is high
FIFO_6	Testing that Empty flag is high when count equal to zero	Randomized testing throughout the simulation	bins for cross coverage between wr_en, rd_en and Empty	Assertion in the design to check that empty is high
FIFO_7	Testing that almostfull flag is high when count equal to FIFO_DEPTH-1	Randomized testing throughout the simulation	bins for cross coverage between wr_en, rd_en & almostfull	Assertion in the design to check that almostfull is high
FIFO_8	Testing that almostempty flag is high when count equal to one	Randomized testing throughout the simulation	bins for cross coverage between wr_en, rd_en & almostempty	Assertion in the design to check that almostempty is high

Verification Diagram: -



HDL (System Verilog) Verification Content: -

Design Module: Design bugs are Explained and Fixed in Comments

D: > new hard > Sessions Digital verification > Sync FIFO project UVM > FIFO.sv

```
module FIFO(FIFO_if.DUT FIFOif);
parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;

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

assign clk = FIFOif.clk;
assign rst_n = FIFOif.rst_n;
assign data_in = FIFOif.data_in;
assign wr_en = FIFOif.wr_en;
assign rd_en = FIFOif.rd_en;
assign FIFOif.data_out=data_out;
assign FIFOif.wr_ack=wr_ack;
assign FIFOif.overflow=overflow;
assign FIFOif.full=full;
assign FIFOif.empty=empty;
assign FIFOif.almostfull=almostfull;
assign FIFOif.almostempty=almostempty;
assign FIFOif.underflow=underflow;

localparam max_fifo_addr = $clog2(FIFO_DEPTH);

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

initial $readmemb("FIFO_init.dat", mem);                                    //Initializing the FIFO to be empty

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;
        underflow <= 0;                                                     //Reseting the Underflow
        overflow <= 0;                                                      //Reseting the Overflow
        wr_ack <=0;
    end
    else if (wr_en && count < FIFO_DEPTH) begin
        mem[wr_ptr] <= data_in;
        wr_ack <= 1;
        wr_ptr <= wr_ptr + 1;
        overflow <= 0;                                                      //Adding condition that if writing then overflow is zero
    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;
        underflow <= 0;                                    //Adding condition that if reading then Underflow is zero
    end
    else begin                                             //Adding the underflow statements to be sequential and not combinational
        if (empty & rd_en)
            underflow <= 1;
        else
            underflow <= 0;
    end
end

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        count <= 0;
    end
    else begin
        if  ( ({wr_en, rd_en} == 2'b10) && !full)
            count <= count + 1;
        else if ( ({wr_en, rd_en} == 2'b01) && !empty)
            count <= count - 1;
        else if (({wr_en, rd_en} == 2'b11}) && full)       //Added conition when FIFO is full and wr and rd enables are high system reads only
            count <= count - 1;
        else if (({wr_en, rd_en} == 2'b11}) && empty)      //Added conition when FIFO is empty and wr and rd enables are high system writes only
            count <= count + 1;
    end
end

assign full = (count == FIFO_DEPTH)? 1 : 0;
assign empty = (count == 0)? 1 : 0;
assign almostfull = (count == FIFO_DEPTH-1)? 1 : 0;        //AlmostFull is when count equal to FIFO_DEPTH-1 Not FIFO_DEPTH-2 as count is incremented on ptr by 1
assign almostempty = (count == 1)? 1 : 0;

endmodule
```

Top Module:

D: > new hard > Sessions Digital verification > Sync FIFO project UVM > FIFO_top.sv

```
import FIFO_test_pkg::*;
import FIFO_env_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
module FIFO_top();
logic clk;
initial begin
    clk=0;
    forever begin
        #1 clk=~clk;
    end
end
FIFO_if FIFOif(clk);
FIFO FIFO_DUT(FIFOif);
bind FIFO FIFO_assertions FIFO_assertions_inst(FIFOif);

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

Interface:

D: > new hard > Sessions Digital verification > Sync FIFO project UVM > FIFO_if.sv

```
interface FIFO_if(clk);
input bit clk;

parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;

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 full, empty, almostfull, almostempty, underflow;

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

endinterface
```

Test Package:

D: > new hard > Sessions Digital verification > Sync FIFO project UVM > FIFO_test.sv

```
package FIFO_test_pkg;
import FIFO_env_pkg::*;
import FIFO_config_pkg::*;
import FIFO_sequence_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class FIFO_test extends uvm_test;

    `uvm_component_utils(FIFO_test)

    virtual FIFO_if FIFO_test_vif;
    FIFO_env env;
    FIFO_config FIFO_config_obj_test;
    FIFO_sequence FIFO_test_seq;

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

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        FIFO_config_obj_test = FIFO_config::type_id::create("FIFO_config_obj_test");
        env = FIFO_env::type_id::create("env",this);
        FIFO_test_seq = FIFO_sequence::type_id::create("FIFO_test_seq",this);

        if(!uvm_config_db #(virtual FIFO_if)::get(this,"","FIFO_IF",FIFO_config_obj_test.FIFO_config_vif))
            `uvm_fatal("build_phase","Test - Unable to get the virtual interface of the FIFO from the uvm_config_db");
        uvm_config_db #(FIFO_config)::set(this,"*","CFG",FIFO_config_obj_test);
    endfunction

```

```
    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        phase.raise_objection(this);
        `uvm_info("run_phase","Stimulus Generation Started",UVM_LOW)
        FIFO_test_seq.start(env.agent_env.sequencer_agent);
        `uvm_info("run_phase","Stimulus Generation Ended",UVM_LOW)
        phase.drop_objection(this);
    endtask: run_phase

endclass: FIFO_test
endpackage
```

Environment Package:

D: > new hard > Sessions Digital verification > Sync FIFO project UVM > FIFO_env.sv

```
package FIFO_env_pkg;
import FIFO_agent_pkg::*;
import FIFO_coverage_pkg::*;
import FIFO_scoreboard_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class FIFO_env extends uvm_env;
`uvm_component_utils(FIFO_env)

FIFO_agent agent_env;
FIFO_coverage coverage_env;
FIFO_scoreboard scoreboard_env;

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

function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    agent_env = FIFO_agent::type_id::create("agent_env",this);
    coverage_env = FIFO_coverage::type_id::create("coverage_env",this);
    scoreboard_env = FIFO_scoreboard::type_id::create("scoreboard_env",this);

endfunction: build_phase

function void connect_phase(uvm_phase phase);
    super.connect_phase(phase);
    agent_env.agent_ap.connect(coverage_env.cov_export);
    agent_env.agent_ap.connect(scoreboard_env.sb_export);
endfunction

endclass
endpackage
```

Agent Package:

D: > new hard > Sessions Digital verification > Sync FIFO project UVM > FIFO_agent_pkg.sv

```
1  package FIFO_agent_pkg;
2  import FIFO_sequenceitem_pkg::*;
3  import FIFO_sequencer_pkg::*;
4  import FIFO_driver_pkg::*;
5  import FIFO_monitor_pkg::*;
6  import FIFO_config_pkg::*;
7  import uvm_pkg::*;
8  `include "uvm_macros.svh"
9
10 class FIFO_agent extends uvm_agent;
11 `uvm_component_utils(FIFO_agent);
12
13 FIFO_config config_agent;
14 FIFO_sequencer sequencer_agent;
15 FIFO_driver driver_agent;
16 FIFO_monitor monitor_agent;
17
18 uvm_analysis_port #(MySequenceItem) agent_ap;
19
20 function new(string name = "FIFO_agent",uvm_component parent = null);
21     super.new(name,parent);
22 endfunction
23
24 function void build_phase(uvm_phase phase);
25     super.build_phase(phase);
26     if(!uvm_config_db #(FIFO_config)::get(this,"","CFG",config_agent))
27         `uvm_fatal("build_phase","Unable to get configruation object");
28
29     sequencer_agent=FIFO_sequencer::type_id::create("sequencer_agent",this);
30     driver_agent=FIFO_driver::type_id::create("driver_agent",this);
31     monitor_agent=FIFO_monitor::type_id::create("monitor_agent",this);
32     agent_ap= new("agent_ap",this);
33 endfunction
34
35 function void connect_phase(uvm_phase phase);
36     driver_agent.FIFO_driver_vif =  config_agent.FIFO_config_vif;
37     monitor_agent.FIFO_monitor_vif = config_agent.FIFO_config_vif;
38     driver_agent.seq_item_port.connect(sequencer_agent.seq_item_export);
39     monitor_agent.monitor_ap.connect(agent_ap);
40 endfunction
41 endclass
42 endpackage
```

Driver Package:

D: > new hard > Sessions Digital verification > Sync FIFO project UVM > FIFO_driver.sv

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

class FIFO_driver extends uvm_driver #(MySequenceItem);

    `uvm_component_utils(FIFO_driver)

    virtual FIFO_if FIFO_driver_vif;
    FIFO_config FIFO_config_obj_driver;
    MySequenceItem item_driver;

    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
            item_driver = MySequenceItem::type_id::create("item_driver");
            seq_item_port.get_next_item(item_driver);

            FIFO_driver_vif.rst_n=item_driver.rst_n;
            FIFO_driver_vif.wr_en=item_driver.wr_en;
            FIFO_driver_vif.rd_en=item_driver.rd_en;
            FIFO_driver_vif.data_in=item_driver.data_in;

            @(negedge FIFO_driver_vif.clk);
            seq_item_port.item_done();
            `uvm_info("run_phase",item_driver.convert2string_stimulus(),UVM_HIGH);
    end
    endtask
endclass
endpackage
```

Sequencer Package:

D: > new hard > Sessions Digital verification > Sync FIFO project UVM > FIFO_sequencer_pkg.sv

```
package FIFO_sequencer_pkg;

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

class FIFO_sequencer extends uvm_sequencer #(MySequenceItem);

`uvm_component_utils(FIFO_sequencer);

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

endclass
endpackage
```

Monitor Module:

D: > new hard > Sessions Digital verification > Sync FIFO project UVM > FIFO_monitor_pkg.sv

```
1   package FIFO_monitor_pkg;
2
3   import FIFO_sequenceitem_pkg::*;
4   import uvm_pkg::*;
5   `include "uvm_macros.svh"
6
7   class FIFO_monitor extends uvm_monitor;
8
9       `uvm_component_utils(FIFO_monitor);
10      virtual FIFO_if FIFO_monitor_vif;
11      MySequenceItem item_monitor;
12      uvm_analysis_port #(MySequenceItem) monitor_ap;
13
14      function new(string name = "FIFO_monitor", uvm_component parent = null);
15          super.new(name , parent);
16      endfunction //new()
17
18      function void build_phase(uvm_phase phase);
19          super.build_phase(phase);
20          monitor_ap = new("monitor_ap",this);
21      endfunction
22
23      task run_phase(uvm_phase phase);
24          super.run_phase(phase);
25          forever begin
26              item_monitor = MySequenceItem::type_id::create("item_monitor");
27              @(negedge FIFO_monitor_vif.clk);
28              item_monitor.rst_n=FIFO_monitor_vif.rst_n;
29              item_monitor.wr_en=FIFO_monitor_vif.wr_en;
30              item_monitor.rd_en=FIFO_monitor_vif.rd_en;
31              item_monitor.data_in=FIFO_monitor_vif.data_in;
32              item_monitor.data_out=FIFO_monitor_vif.data_out;
33              item_monitor.wr_ack=FIFO_monitor_vif.wr_ack;
34              item_monitor.overflow=FIFO_monitor_vif.overflow;
35              item_monitor.underflow=FIFO_monitor_vif.underflow;
36              item_monitor.full=FIFO_monitor_vif.full;
37              item_monitor.empty=FIFO_monitor_vif.empty;
38              item_monitor.almostfull=FIFO_monitor_vif.almostfull;
39              item_monitor.almostempty=FIFO_monitor_vif.almostempty;
40
41              monitor_ap.write(item_monitor);
42              `uvm_info("run_phase",item_monitor.convert2string(),UVM_HIGH)
43          end
44      endtask
45  endclass //FIFO_monitor extends superClass
46  endpackage
```

Sequence item Package:

D: > new hard > Sessions Digital verification > Sync FIFO project UVM > FIFO_sequenceitem_pkg.sv

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

class MySequenceItem extends uvm_sequence_item;
`uvm_object_utils(MySequenceItem)

//Signals Declarations

rand logic rst_n, wr_en, rd_en;
rand logic [15:0] data_in;

logic [15:0] data_out;
logic clk,wr_ack, overflow,full, empty, almostfull, almostempty, underflow;



//Constructor for the sequence item
function new(string name = "MySequenceItem");
    super.new(name);

endfunction

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

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

//Constraint blocks
constraint reset{rst_n dist {0:=5,1:=95};}
constraint write_enable{wr_en dist{0:=(30),1:=70};}
constraint read_enable{rd_en dist{0:=(70),1:=30};}

endclass
endpackage
```

Sequence Package:

D: > new hard > Sessions Digital verification > Sync FIFO project UVM > FIFO_sequence_pkg.sv

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

class FIFO_sequence extends uvm_sequence #(MySequenceItem);

`uvm_object_utils(FIFO_sequence);
MySequenceItem item;

//Constructor for the sequence
function new(string name = "MySequence");
    super.new(name);
endfunction

//Main task for the sequence
virtual task body();
    //Reset initialization
    item = MySequenceItem::type_id::create("item"); //Create a sequence item
    start_item(item);
    item.rst_n=0; item.data_in=0; item.wr_en=0; item.rd_en=0;
    finish_item(item);


    repeat(10000) begin
        item = MySequenceItem::type_id::create("item"); //Create a sequence item
        start_item(item);
        assert (item.randomize());
        finish_item(item);
    end
endtask
endclass
endpackage
```

Configuration object Package:

D: > new hard > Sessions Digital verification > Sync FIFO project UVM > FIFO_config.sv

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

    class FIFO_config extends uvm_object;

        `uvm_object_utils(FIFO_config)

        virtual FIFO_if FIFO_config_vif;

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

Functional Coverage Package:

D: > new hard > Sessions Digital verification > Sync FIFO project UVM > FIFO_coverage_pkg.sv

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

class FIFO_coverage extends uvm_component;
`uvm_component_utils(FIFO_coverage);

uvm_analysis_export #(MySequenceItem) cov_export;
uvm_tlm_analysis_fifo #(MySequenceItem) cov_fifo;
MySequenceItem item_cov;

//Covergroups
covergroup cov_gp ;

    data_out_cp: coverpoint item_cov.data_out {
        bins data_out_bin = {[0:$]};
        option.weight=0;
        }

    wr_rd_data_out: cross item_cov.wr_en,item_cov.rd_en,data_out_cp;
    wr_rd_wr_ack: cross item_cov.wr_en,item_cov.rd_en,item_cov.wr_ack;
    wr_rd_overflow: cross item_cov.wr_en,item_cov.rd_en,item_cov.overflow;
    wr_rd_underflow: cross item_cov.wr_en,item_cov.rd_en,item_cov.underflow;
    wr_rd_full: cross item_cov.wr_en,item_cov.rd_en,item_cov.full;
    wr_rd_empty: cross item_cov.wr_en,item_cov.rd_en,item_cov.empty;
    wr_rd_almostfull: cross item_cov.wr_en,item_cov.rd_en,item_cov.almostfull;
    wr_rd_almostempty: cross item_cov.wr_en,item_cov.rd_en,item_cov.almostempty;
    //option.auto_bin_max=0;

endgroup

//Defining the new function and the covergroup inside it
function new(string name = "FIFO_coverage", uvm_component parent = null);
    super.new(name,parent);
    cov_gp=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(item_cov);
        cov_gp.sample();
    end
endtask
endclass
endpackage
```

Scoreboard Package: (Reference model)

D: > new hard > Sessions Digital verification > Sync FIFO project UVM > FIFO_scoreboard_pkg.sv

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

class FIFO_scoreboard extends uvm_scoreboard;

    `uvm_component_utils(FIFO_scoreboard);
    uvm_analysis_export #(MySequenceItem) sb_export;
    uvm_tlm_analysis_fifo #(MySequenceItem) sb_fifo;

    //Signals declaration
    MySequenceItem item_scoreboard;
    logic [15:0] data_out_ref;
    logic wr_ack_ref, overflow_ref,underflow_ref,full_ref, empty_ref, almostfull_ref, almostempty_ref;
    int Error_count=0, Correct_count=0, count=0;

    //FIFO queue
    logic [15:0] FIFO_queue [$];

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

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

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

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever begin
            sb_fifo.get(item_scoreboard);
            ref_model(item_scoreboard);
            if((item_scoreboard.data_out!=data_out_ref) || (item_scoreboard.wr_ack!=wr_ack_ref) || (item_scoreboard.overflow!=overflow_ref) ||
            (item_scoreboard.underflow!=underflow_ref) || (item_scoreboard.full!=full_ref) || (item_scoreboard.empty!=empty_ref) ||
            (item_scoreboard.almostfull!=almostfull_ref) || (item_scoreboard.almostempty!= almostempty_ref)) begin
                Error_count++;
                `uvm_error("run_phase",
                $sformatf("Comparison failed, Transaction received by DUT: %s while reference outputs: data_out_ref=0b%0b, wr_ack_ref=0b%0b,
                overflow_ref=0b%0b, underflow_ref=0b%0b, full_ref=0b%0b, empty_ref=0b%0b, almostfull_ref=0b%0b, almostempty_ref=0b%0b",
                item_scoreboard.convert2string(), data_out_ref, wr_ack_ref, overflow_ref, underflow_ref, full_ref, empty_ref, almostfull_ref, almostempty_ref)
);
            end
            else
                Correct_count++;
        end
    endtask

    task ref_model(MySequenceItem item_check);
        ////////////////////////////////Reset Signal//////////////////////////////
        if(!item_check.rst_n) begin
            FIFO_queue.delete();
            count=0; wr_ack_ref=0; overflow_ref=0; underflow_ref=0; full_ref=0; empty_ref=1; almostfull_ref=0; almostempty_ref=0;
        end
        else begin

```

```
        ////////////////////////////////Reading and Writing//////////////////////////////
            //write only
            if(item_check.rd_en && item_check.wr_en && count==0) begin
                underflow_ref=1;
                FIFO_queue.push_front(item_check.data_in);
                wr_ack_ref=1;
                overflow_ref=0;
                almostempty_ref=1;
                almostfull_ref=0;
                empty_ref=0;
                full_ref=0;

            count++;
            end
            //Read only
            else if(item_check.rd_en && item_check.wr_en && count==8) begin
                overflow_ref=1;
                data_out_ref=FIFO_queue.pop_back();
                wr_ack_ref=0;
                underflow_ref=0;
                almostempty_ref=0;
                almostfull_ref=1;
                empty_ref=0;
                full_ref=0;

            count--;
            end
            //Write and Read Together
            else begin
                if(item_check.wr_en && count < 8) begin
                    FIFO_queue.push_front(item_check.data_in);
                    count++;
                end
                if(item_check.rd_en && (count != 0)) begin
                    data_out_ref=FIFO_queue.pop_back();
                    count--;
                end

            //All Control signals
            ////////////////////////////////Over and Under flow Signals//////////////////////////////
            //Overflow signal
            if(item_check.wr_en && full_ref)  overflow_ref=1;
            else    overflow_ref=0;
            //Underflow signal
            if(item_check.rd_en && empty_ref)  underflow_ref=1;
            else    underflow_ref=0;
            ////////////////////////////////Write acknowledge Signal//////////////////////////////
            if((item_check.rd_en && item_check.wr_en && empty_ref)||(item_check.wr_en && count <= 8 && overflow_ref==0)) wr_ack_ref=1;
            else wr_ack_ref=0;
            ////////////////////////////////full and empty  Signals//////////////////////////////
            //Full signal
            if(count == 8)  full_ref=1;
            else    full_ref =0;
            //Empty signal
            if(count == 0)  empty_ref=1;
            else    empty_ref=0;
            ////////////////////////////////Almost empty and full Signals//////////////////////////////
            //Almost empty signal
            if(count == 1)    almostempty_ref=1;
            else    almostempty_ref=0;
            //Almost full signal
            if(count==(7)) almostfull_ref=1;
            else    almostfull_ref=0;

            end
        end
    endtask

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

Assertions Module:

D: > new hard > Sessions Digital verification > Sync FIFO project UVM > FIFO_assertions.sv

```
module FIFO_assertions(FIFO_if.DUT FIFOif);

//////////////////////////////Assertions//////////////////////////////
property reset_p;
    @(posedge FIFOif.clk)
    !FIFOif.rst_n |-> (FIFO_DUT.count==0 && !FIFOif.overflow && !FIFOif.underflow && !FIFOif.full && FIFOif.empty && !FIFOif.almostfull && !FIFOif.almostempty && !FIFOif.wr_ack);
endproperty

property overflow_p;
    @(posedge FIFOif.clk)
    disable iff (!FIFOif.rst_n) (FIFOif.wr_en && FIFO_DUT.count==8) |=> (FIFOif.overflow && !FIFOif.wr_ack);
endproperty

property underflow_p;
    @(posedge FIFOif.clk)
    disable iff (!FIFOif.rst_n) (FIFOif.rd_en && FIFO_DUT.count==0) |=> (FIFOif.underflow);
endproperty

property wr_ack_p;
    @(posedge FIFOif.clk)
    disable iff (!FIFOif.rst_n) (FIFO_DUT.count!=8 && FIFOif.wr_en) |=> FIFOif.wr_ack;
endproperty

property full_p;
    @(posedge FIFOif.clk)
    disable iff (!FIFOif.rst_n) (FIFO_DUT.count==8) |-> FIFOif.full;
endproperty

property empty_p;
    @(posedge FIFOif.clk)
    disable iff (!FIFOif.rst_n) (FIFO_DUT.count==0) |-> FIFOif.empty;
endproperty

property almostfull_p;
    @(posedge FIFOif.clk)
    disable iff (!FIFOif.rst_n) (FIFO_DUT.count==7) |-> FIFOif.almostfull;
endproperty

property almostempty_p;
    @(posedge FIFOif.clk)
    disable iff (!FIFOif.rst_n) (FIFO_DUT.count==1) |-> FIFOif.almostempty;
endproperty

reset_assertion: assert property(reset_p);
overflow_assertion: assert property(overflow_p);
underflow_assertion: assert property(underflow_p);
wr_ack_assertion: assert property(wr_ack_p);
full_assertion: assert property(full_p);
empty_assertion: assert property(empty_p);
almostfull_assertion: assert property(almostfull_p);
almostempty_assertion: assert property(almostempty_p);

reset_coverage: cover property(reset_p);
overflow_coverage: cover property(overflow_p);
underflow_coverage: cover property(underflow_p);
wr_ack_coverage: cover property(wr_ack_p);
full_coverage: cover property(full_p);
empty_coverage: cover property(empty_p);
almostfull_coverage: cover property(almostfull_p);
almostempty_coverage: cover property(almostempty_p);

endmodule
```

Do file:

Excluding invalid cases to happen from the coverage as wr_en=1, rd_en=0, empty=1

D: > new hard > Sessions Digital verification > Sync FIFO project UVM > run.do

```
vlib work
vlog -f src_files.list.txt +cover -covercells
vsim -voptargs=+acc work.FIFO_top -classdebug -uvmcontrol=all -cover
add wave /FIFO_top/FIFOif/*
add wave -position insertpoint  \
sim:/FIFO_top/FIFO_DUT/mem \
sim:/FIFO_top/FIFO_DUT/wr_ptr \
sim:/FIFO_top/FIFO_DUT/rd_ptr \
sim:/FIFO_top/FIFO_DUT/count
run -all
coverage save FIFO_top.ucdb -onexit
coverage exclude -cvgpath
{/FIFO_coverage_pkg/FIFO_coverage/cov_gp/wr_rd_wr_ack/<auto[0],auto[1],auto[1]>}
{/FIFO_coverage_pkg/FIFO_coverage/cov_gp/wr_rd_wr_ack/<auto[0],auto[0],auto[1]>}
{/FIFO_coverage_pkg/FIFO_coverage/cov_gp/wr_rd_overflow/<auto[0],auto[1],auto[1]>}
{/FIFO_coverage_pkg/FIFO_coverage/cov_gp/wr_rd_overflow/<auto[0],auto[0],auto[1]>}
{/FIFO_coverage_pkg/FIFO_coverage/cov_gp/wr_rd_underflow/<auto[1],auto[0],auto[1]>}
{/FIFO_coverage_pkg/FIFO_coverage/cov_gp/wr_rd_underflow/<auto[0],auto[0],auto[1]>}
{/FIFO_coverage_pkg/FIFO_coverage/cov_gp/wr_rd_full/<auto[1],auto[1],auto[1]>}
{/FIFO_coverage_pkg/FIFO_coverage/cov_gp/wr_rd_full/<auto[0],auto[1],auto[1]>}
```

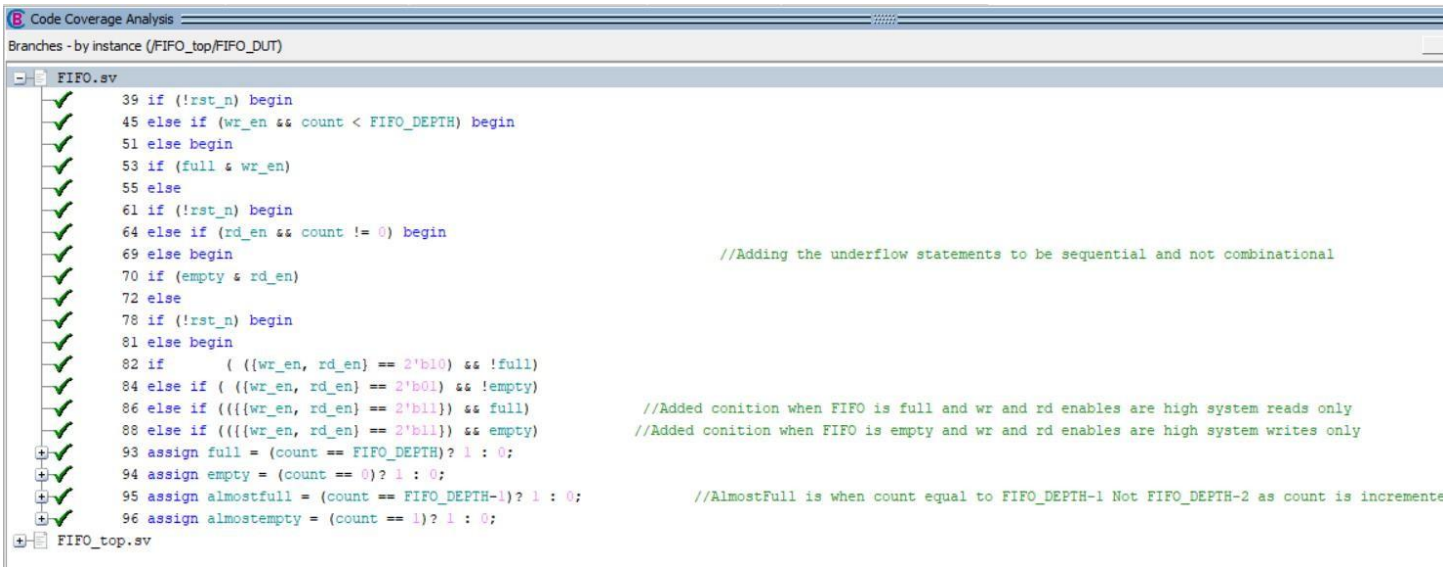
Src_files.list:

D: > new hard > Sessions Digital verification > Sync FIFO project UVM > src_files.list.txt

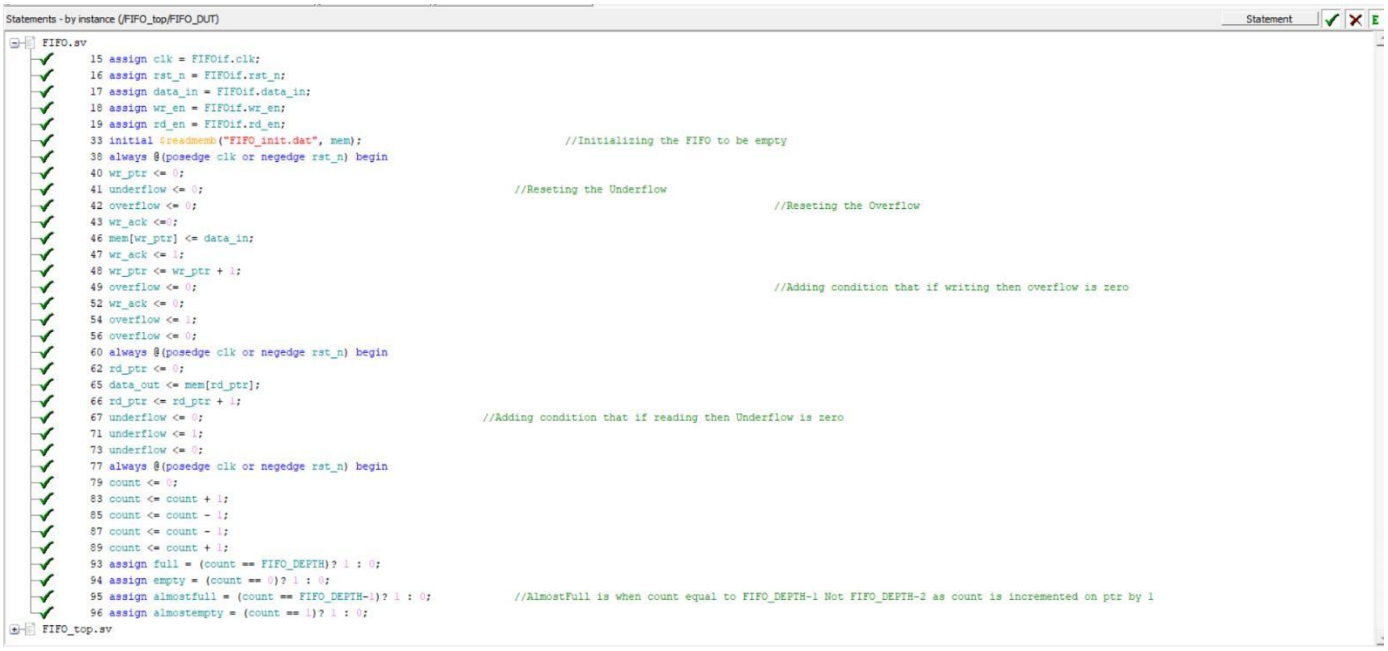
```
FIFO_if.sv
FIFO.sv
FIFO_sequenceitem_pkg.sv
FIFO_sequence_pkg.sv
FIFO_sequencer_pkg.sv
FIFO_config.sv
FIFO_monitor_pkg.sv
FIFO_driver.sv
FIFO_agent_pkg.sv
FIFO_coverage_pkg.sv
FIFO_scoreboard_pkg.sv
FIFO_env.sv
FIFO_test.sv
FIFO_assertions.sv
FIFO_top.sv
```

Code Coverage: -

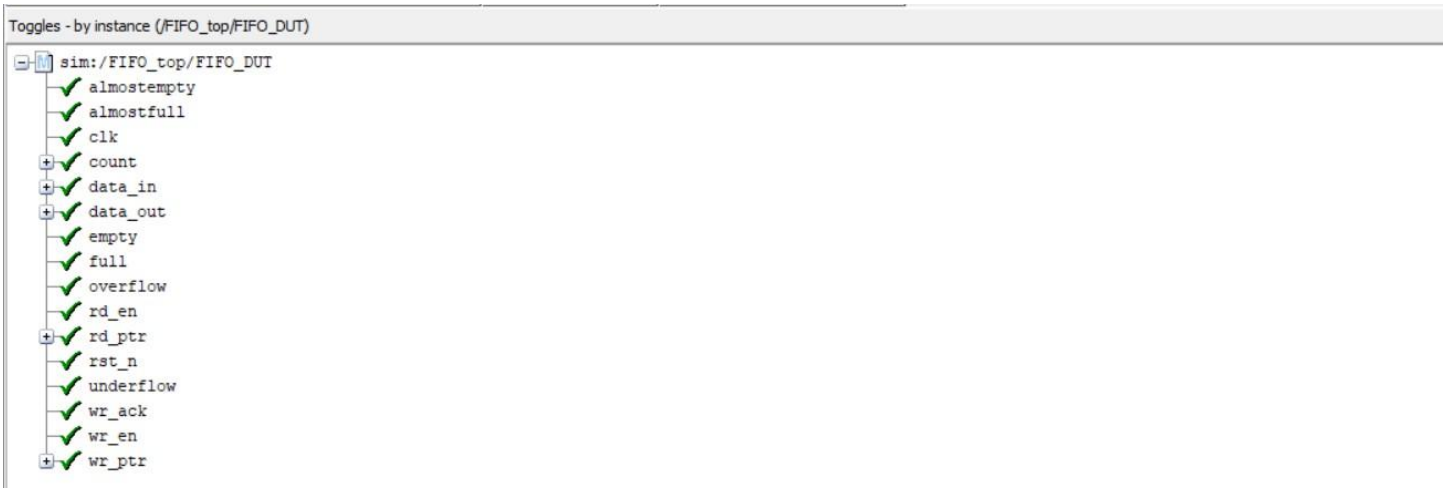
Branch Coverage:



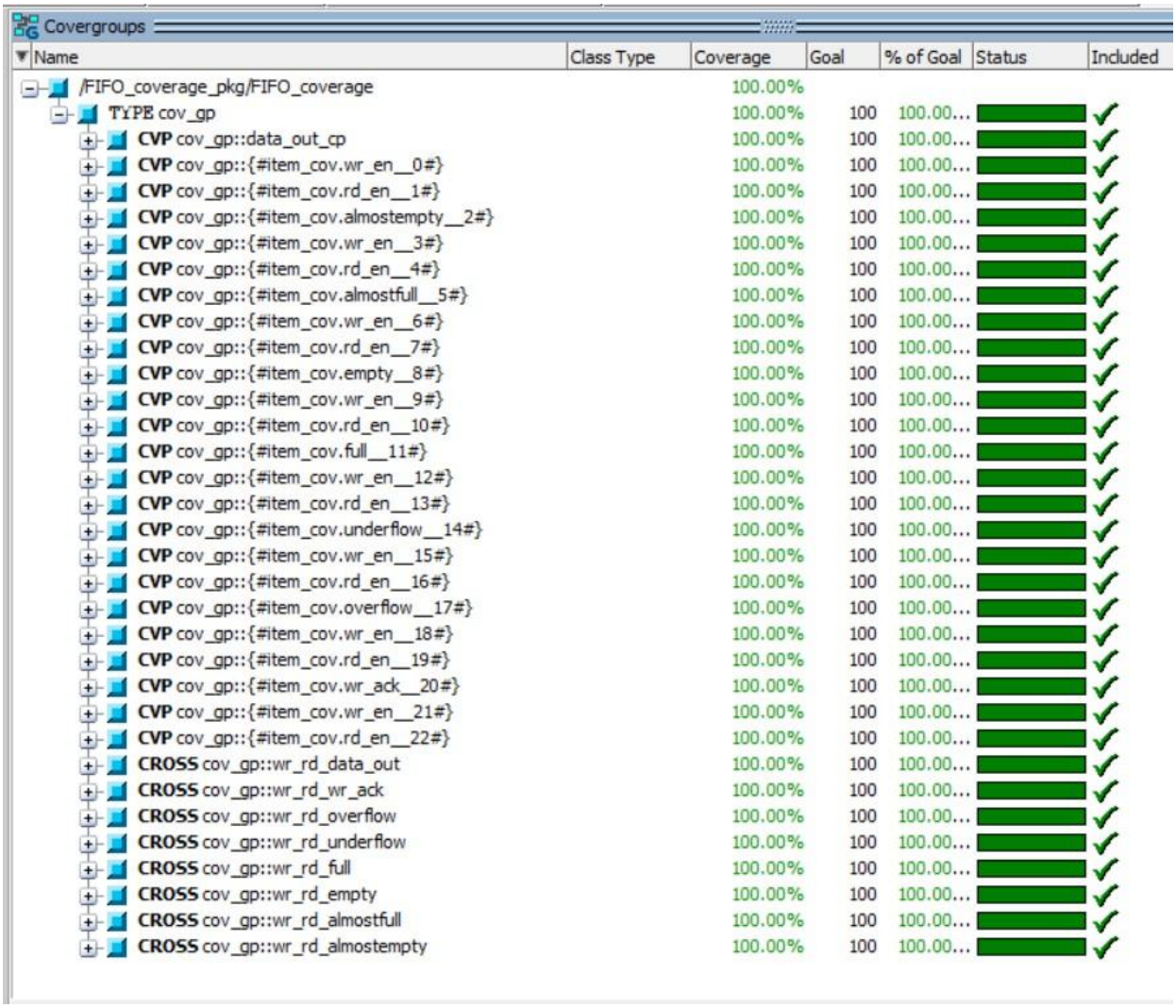
Statement Coverage:



Toggle Coverage:



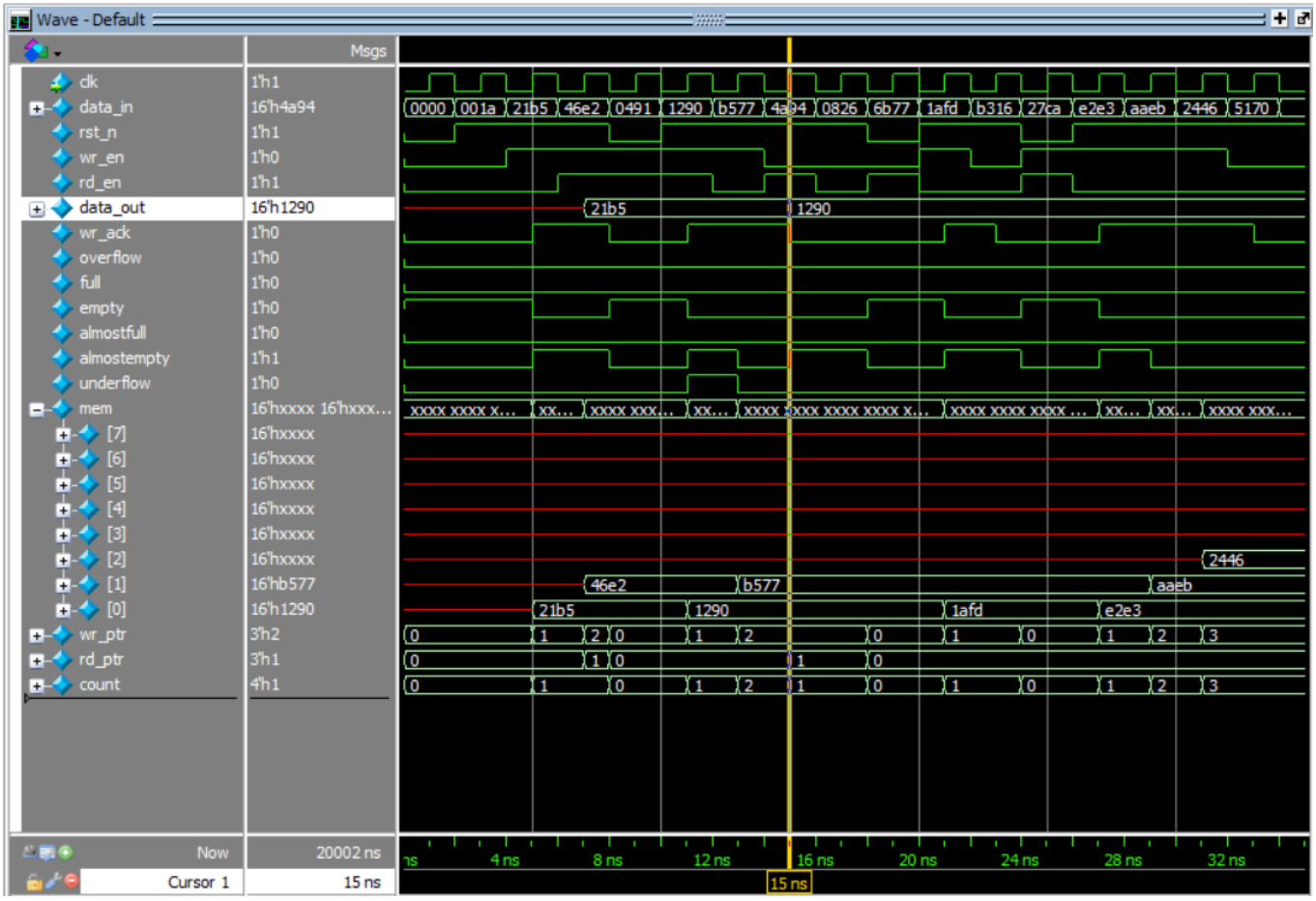
Functional Coverage: -



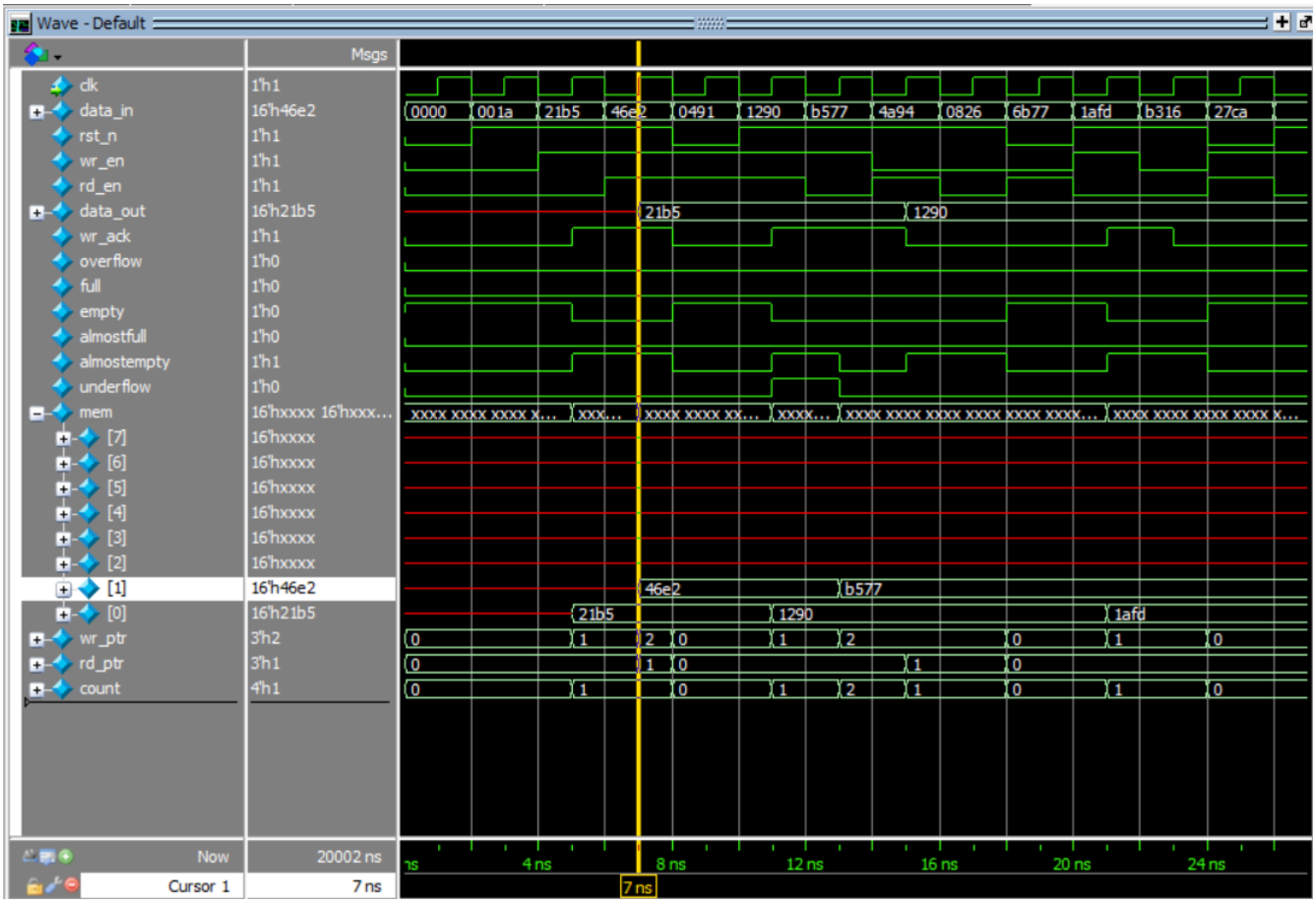
Name	Class Type	Coverage	Goal	% of Goal	Status	Included
/FIFO_coverage_pkg/FIFO_coverage		100.00%				
TYPE cov_gp		100.00%	100	100.00...		
CVP cov_gp::data_out_cp		100.00%	100	100.00...		
CVP cov_gp::{#item_cov.wr_en__0#}		100.00%	100	100.00...		
CVP cov_gp::{#item_cov.rd_en__1#}		100.00%	100	100.00...		
CVP cov_gp::{#item_cov.almostempty__2#}		100.00%	100	100.00...		
CVP cov_gp::{#item_cov.wr_en__3#}		100.00%	100	100.00...		
CVP cov_gp::{#item_cov.rd_en__4#}		100.00%	100	100.00...		
CVP cov_gp::{#item_cov.almostfull__5#}		100.00%	100	100.00...		
CVP cov_gp::{#item_cov.wr_en__6#}		100.00%	100	100.00...		
CVP cov_gp::{#item_cov.rd_en__7#}		100.00%	100	100.00...		
CVP cov_gp::{#item_cov.empty__8#}		100.00%	100	100.00...		
CVP cov_gp::{#item_cov.wr_en__9#}		100.00%	100	100.00...		
CVP cov_gp::{#item_cov.rd_en__10#}		100.00%	100	100.00...		
CVP cov_gp::{#item_cov.full__11#}		100.00%	100	100.00...		
CVP cov_gp::{#item_cov.wr_en__12#}		100.00%	100	100.00...		
CVP cov_gp::{#item_cov.rd_en__13#}		100.00%	100	100.00...		
CVP cov_gp::{#item_cov.underflow__14#}		100.00%	100	100.00...		
CVP cov_gp::{#item_cov.wr_en__15#}		100.00%	100	100.00...		
CVP cov_gp::{#item_cov.rd_en__16#}		100.00%	100	100.00...		
CVP cov_gp::{#item_cov.overflow__17#}		100.00%	100	100.00...		
CVP cov_gp::{#item_cov.wr_en__18#}		100.00%	100	100.00...		
CVP cov_gp::{#item_cov.rd_en__19#}		100.00%	100	100.00...		
CVP cov_gp::{#item_cov.wr_ack__20#}		100.00%	100	100.00...		
CVP cov_gp::{#item_cov.wr_en__21#}		100.00%	100	100.00...		
CVP cov_gp::{#item_cov.rd_en__22#}		100.00%	100	100.00...		
CROSS cov_gp::wr_rd_data_out		100.00%	100	100.00...		
CROSS cov_gp::wr_rd_wr_ack		100.00%	100	100.00...		
CROSS cov_gp::wr_rd_overflow		100.00%	100	100.00...		
CROSS cov_gp::wr_rd_underflow		100.00%	100	100.00...		
CROSS cov_gp::wr_rd_full		100.00%	100	100.00...		
CROSS cov_gp::wr_rd_empty		100.00%	100	100.00...		
CROSS cov_gp::wr_rd_almostfull		100.00%	100	100.00...		
CROSS cov_gp::wr_rd_almostempty		100.00%	100	100.00...		

Quest-Sim Snippets: -

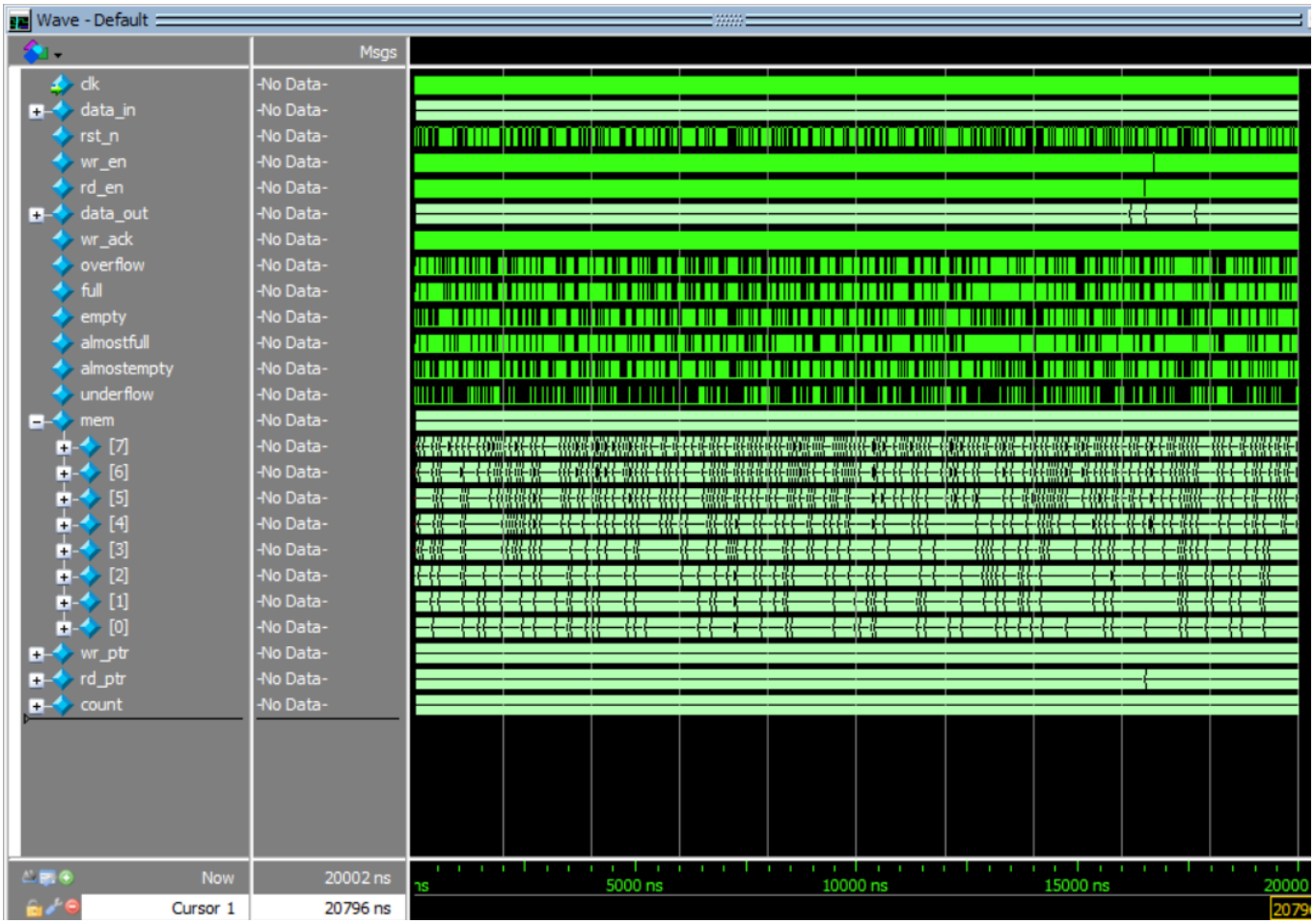
Read Test case:



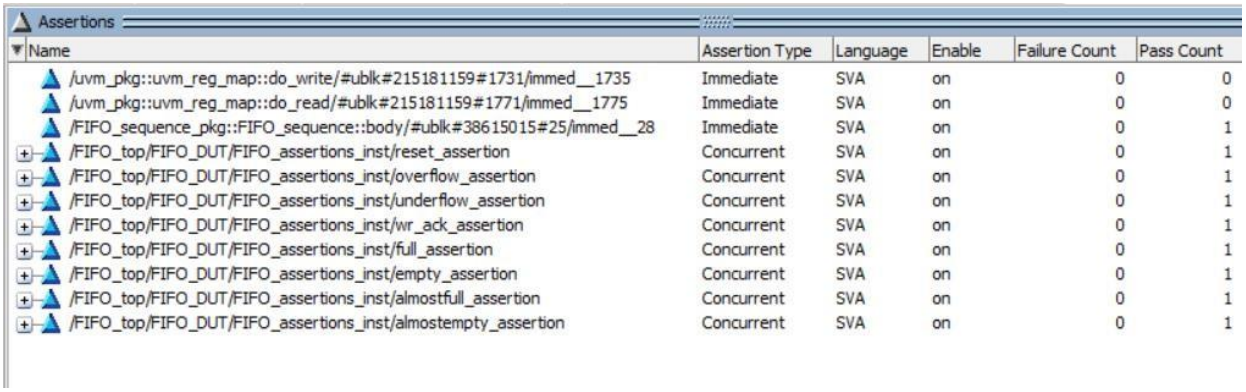
Write Test case:



All iterations Snippet:



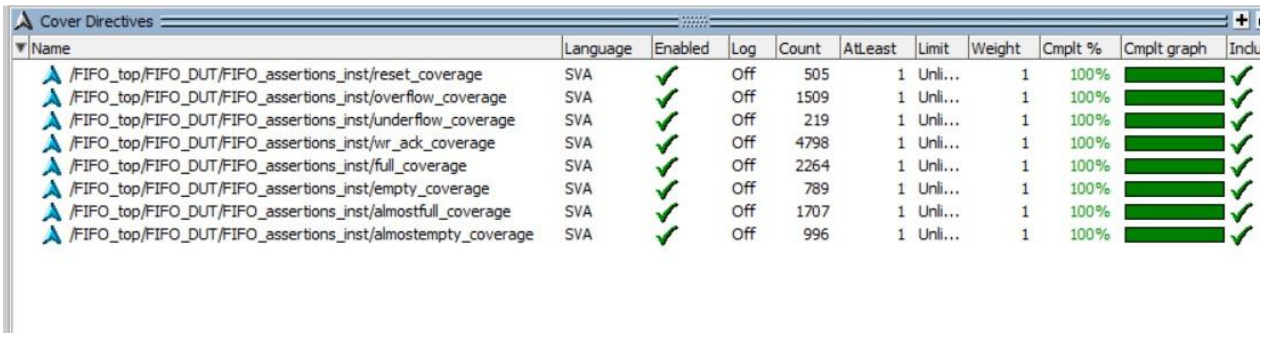
Assertions:



Assertions

Name	Assertion Type	Language	Enable	Failure Count	Pass Count
/uvm_pkg::uvm_reg_map::do_write/#ublk#215181159#1731/immed__1735	Immediate	SVA	on	0	0
/uvm_pkg::uvm_reg_map::do_read/#ublk#215181159#1771/immed__1775	Immediate	SVA	on	0	0
/FIFO_sequence_pkg::FIFO_sequence::body/#ublk#38615015#25/immed__28	Immediate	SVA	on	0	1
/FIFO_top/FIFO_DUT/FIFO_assertions_inst/reset_assertion	Concurrent	SVA	on	0	1
/FIFO_top/FIFO_DUT/FIFO_assertions_inst/overflow_assertion	Concurrent	SVA	on	0	1
/FIFO_top/FIFO_DUT/FIFO_assertions_inst/underflow_assertion	Concurrent	SVA	on	0	1
/FIFO_top/FIFO_DUT/FIFO_assertions_inst/wr_ack_assertion	Concurrent	SVA	on	0	1
/FIFO_top/FIFO_DUT/FIFO_assertions_inst/full_assertion	Concurrent	SVA	on	0	1
/FIFO_top/FIFO_DUT/FIFO_assertions_inst/empty_assertion	Concurrent	SVA	on	0	1
/FIFO_top/FIFO_DUT/FIFO_assertions_inst/almostfull_assertion	Concurrent	SVA	on	0	1
/FIFO_top/FIFO_DUT/FIFO_assertions_inst/almostempty_assertion	Concurrent	SVA	on	0	1

Cover Directives:



Cover Directives

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Inclu
/FIFO_top/FIFO_DUT/FIFO_assertions_inst/reset_coverage	SVA	✓	Off	505	1	Unli...	1	100%		✓
/FIFO_top/FIFO_DUT/FIFO_assertions_inst/overflow_coverage	SVA	✓	Off	1509	1	Unli...	1	100%		✓
/FIFO_top/FIFO_DUT/FIFO_assertions_inst/underflow_coverage	SVA	✓	Off	219	1	Unli...	1	100%		✓
/FIFO_top/FIFO_DUT/FIFO_assertions_inst/wr_ack_coverage	SVA	✓	Off	4798	1	Unli...	1	100%		✓
/FIFO_top/FIFO_DUT/FIFO_assertions_inst/full_coverage	SVA	✓	Off	2264	1	Unli...	1	100%		✓
/FIFO_top/FIFO_DUT/FIFO_assertions_inst/empty_coverage	SVA	✓	Off	789	1	Unli...	1	100%		✓
/FIFO_top/FIFO_DUT/FIFO_assertions_inst/almostfull_coverage	SVA	✓	Off	1707	1	Unli...	1	100%		✓
/FIFO_top/FIFO_DUT/FIFO_assertions_inst/almostempty_coverage	SVA	✓	Off	996	1	Unli...	1	100%		✓

Transcript:

Error Count =0 and Correct Count = 10,001

```
#    Time: 0 ns  Iteration: 0  Instance: /FIFO_top/FIFO_DUT
# UVM_INFO @ 0: reporter [RNTST] Running test FIFO_test...
# ** Warning: In instance '\/FIFO_coverage_pkg::FIFO_coverage::cov_gp ' the 'option.per_instance' is set to '0' (false). Assignment to members 'weight', 'goal' and 'comment' of 'option' would not have any effect.
# UVM_INFO FIFO_test.sv(34) @ 0: uvm_test_top [run_phase] Stimulus Generation Started
# ****************************************************************************
# * Questa UVM Transaction Recording Turned ON.                              *
# * recording_detail has been set.                                           *
# *  To turn off, set 'recording_detail' to off:                             *
# * uvm_config_db#(int)            ::set(null, "", "recording_detail", 0); *
# * uvm_config_db#(uvm_bitstream_t)::set(null, "", "recording_detail", 0); *
# ****************************************************************************
# UVM_INFO FIFO_test.sv(36) @ 20002: uvm_test_top [run_phase] Stimulus Generation Ended
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 20002: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO FIFO_scoreboard_pkg.sv(134) @ 20002: uvm_test_top.env.scoreboard_env [report_phase] Total successful transactions: 10001
# UVM_INFO FIFO_scoreboard_pkg.sv(135) @ 20002: uvm_test_top.env.scoreboard_env [report_phase] Total failed transactions: 0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :    8
# 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]     2
# ** Note: $finish    : C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 20002 ns  Iteration: 61  Instance: /FIFO_top
# Break in Task uvm_pkg/uvm_root::run_test at C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh line 430
# Causality operation skipped due to absence of debug database file
```