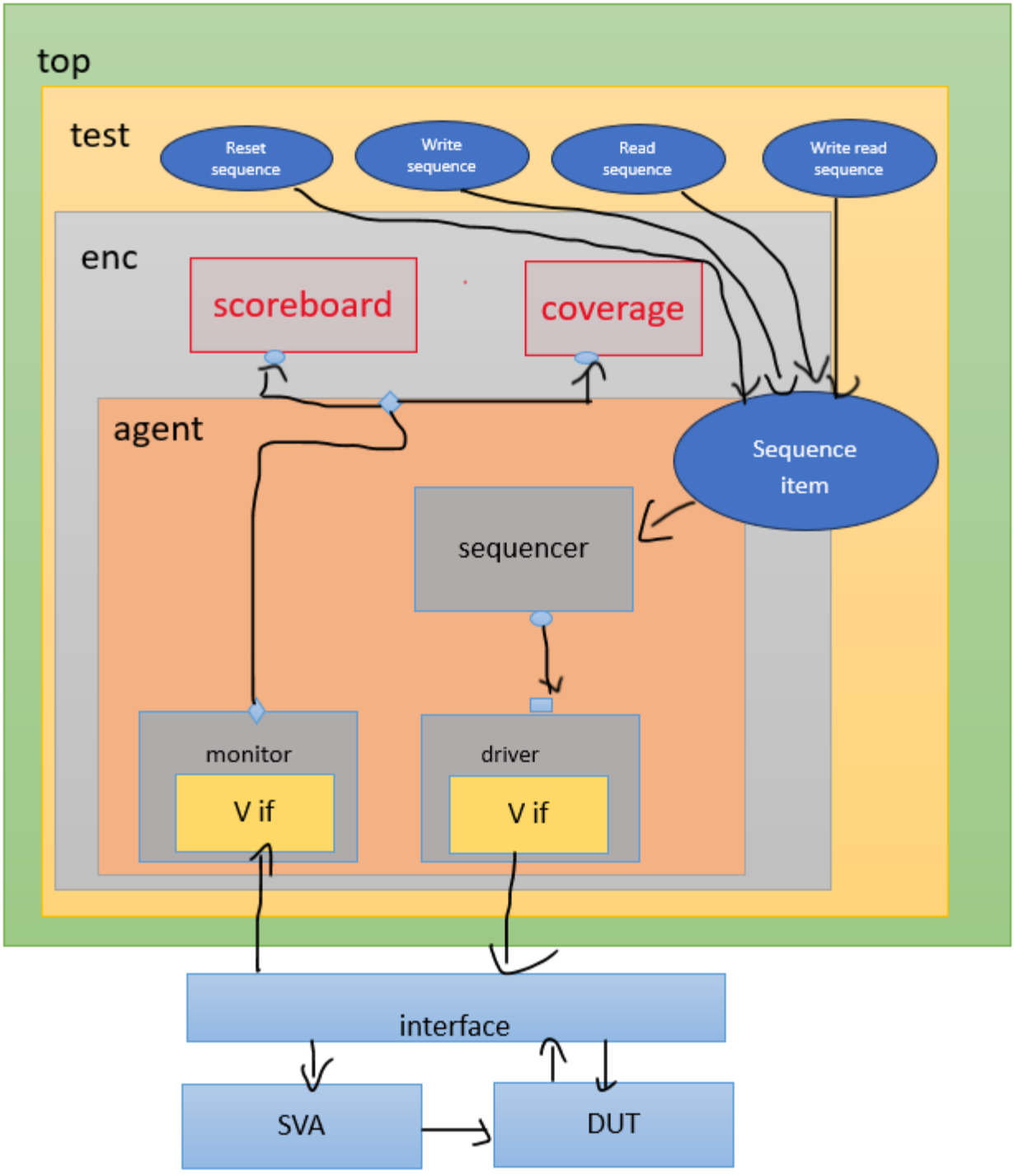
Momen Mostafa Mohamed Elzaghawy

FIFO_UVM_project

Verification plan:

A	B	C	D	E
label	design requirement description	stimulus generation	functional coverage	functionality check
FIFO_1	the rst_n is asserted to check the reset functionality	directed and randomization		assertion to check rst_n functionality
FIFO_2	the rst_n is desserted and wr_en is asserted to check write functionality	randomization	cross coverage	assertion to check write functionality
FIFO_3	the rst_n is desserted and rd_en is asserted to check read functionality	randomization	cross coverage	assertion to check read functionality
FIFO_4	the rst_n is desserted and rd_en =1,wr_en=1 to check the functionality	randomization	cross coverage	assertion to check read functionality

UVM_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 4 sequences: Reset, Write only, Read only, Write Read .

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.

Assertions table:

	A	B
	feature	assertion
	whenever the rst_n is asserted the count and wr_ptr and rd_ptr = 0	@(posedge fifoif.clk) !fifoif.rst_n |=> $past(!FIFO.wr_ptr) && $past(!FIFO.rd_ptr) && FIFO.count == 0;
	whenever the rst_n is deasserted , wr_en is 1 , full is 0 then wr_ack is asserted	@(posedge fifoif.clk) disable iff (!fifoif.rst_n) fifoif.wr_en && !fifoif.full |=> fifoif.wr_ack;
	whenever the rst_n is deasserted , wr_en is 1 , full is 1 then overflow is asserted	@(posedge fifoif.clk) disable iff (!fifoif.rst_n) fifoif.wr_en && fifoif.full |=> fifoif.overflow;
	whenever the rst_n is deasserted , rd_en is 1 , empty is 1 then underflow is asserted	@(posedge fifoif.clk) disable iff (!fifoif.rst_n) fifoif.rd_en && fifoif.empty |=> fifoif.underflow;
	whenever the rst_n is deasserted , count=0 then empty is asserted	@(posedge fifoif.clk) disable iff (!fifoif.rst_n) FIFO.count==0 |=> $past(fifoif.empty);
	whenever the rst_n is deasserted , count=FIFO_DEPTH then full is asserted	@(posedge fifoif.clk) disable iff (!fifoif.rst_n) FIFO.count==fifoif.FIFO_DEPTH |=> $past(fifoif.full);
	whenever the rst_n is deasserted , count=FIFO_DEPTH-1 then almostfull is asserted	@(posedge fifoif.clk) disable iff (!fifoif.rst_n) FIFO.count==fifoif.FIFO_DEPTH-1 |=> $past(fifoif.almostfull);
9	whenever the rst_n is deasserted , count=1 then almostempty is asserted	@(posedge fifoif.clk) disable iff (!fifoif.rst_n) FIFO.count==1 |=> $past(fifoif.almostempty);
10	whenever the rst_n is deasserted , wr_en is 1 , full is 0 , wr_ptr=FIFO_DEPTH-1 then wr_ptr is asserted	@(posedge fifoif.clk) disable iff (!fifoif.rst_n) fifoif.wr_en && !fifoif.full && FIFO.wr_ptr==fifoif.FIFO_DEPTH-1 |=> FIFO.wr_ptr==0 ;
11	whenever the rst_n is deasserted , rd_en is 1 , empty is 0 , rd_ptr=FIFO_DEPTH-1 then rd_ptr is asserted	@(posedge fifoif.clk) disable iff (!fifoif.rst_n) fifoif.rd_en && !fifoif.empty && FIFO.rd_ptr==fifoif.FIFO_DEPTH-1 |=> FIFO.rd_ptr==0 ;
12	at any time the wr_ptr <= FIFO_DEPTH-1 , rd_ptr <= FIFO_DEPTH-1 , count <= FIFO_DEPTH	@(posedge fifoif.clk) disable iff (!fifoif.rst_n) FIFO.wr_ptr<=fifoif.FIFO_DEPTH-1 && FIFO.rd_ptr<=fifoif.FIFO_DEPTH-1 && FIFO.count<=fifoif.FIFO_DEPTH ;

Bugs:

1- At rst_n is asserted overflow and underflow and wr_ack is deasserted
2- In line 35 full && wr_en
3- The logic when wr_en and rd_en is asserted is missing
4- underflow logic is not right
5- in line 67 almost full logic is not right (FIFO_DEPTH-1)
6- overflow must be 0 when successful write , underflow must be 0 when successful read

Design (with bugs):

```
8   module FIFO(data_in, wr_en, rd_en, clk, rst_n, full, empty, almostfull, almostempty, wr_ack, overflow, underflow, data_out);
9   parameter FIFO_WIDTH = 16;
10  parameter FIFO_DEPTH = 8;
11  input [FIFO_WIDTH-1:0] data_in;
12  input clk, rst_n, wr_en, rd_en;
13  output reg [FIFO_WIDTH-1:0] data_out;
14  output reg wr_ack, overflow;
15  output full, empty, almostfull, almostempty, underflow;
16
17  localparam max_fifo_addr = $clog2(FIFO_DEPTH);
18
19  reg [FIFO_WIDTH-1:0] mem [FIFO_DEPTH-1:0];
20
21  reg [max_fifo_addr-1:0] wr_ptr, rd_ptr;
22  reg [max_fifo_addr:0] count;
23
24  always @(posedge clk or negedge rst_n) begin
25      if (!rst_n) begin
26          wr_ptr <= 0;
27      end
28      else if (wr_en && count < FIFO_DEPTH) begin
29          mem[wr_ptr] <= data_in;
30          wr_ack <= 1;
31          wr_ptr <= wr_ptr + 1;
32      end
33      else begin
34          wr_ack <= 0;
35          if (full & wr_en)
36              overflow <= 1;
37          else
38              overflow <= 0;
39      end
40  end
41
42  always @(posedge clk or negedge rst_n) begin
43      if (!rst_n) begin
44          rd_ptr <= 0;
45      end
46      else if (rd_en && count != 0) begin
47          data_out <= mem[rd_ptr];
48          rd_ptr <= rd_ptr + 1;
49      end
50  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 (with no bugs):

```
8   module FIFO(FIFO_if.DUT fifoif);
9
10
11  localparam max_fifo_addr = $clog2(fifoif.FIFO_DEPTH);
12
13  reg [fifoif.FIFO_WIDTH-1:0] mem [fifoif.FIFO_DEPTH-1:0];
14
15  reg [max_fifo_addr-1:0] wr_ptr, rd_ptr;
16  reg [max_fifo_addr:0] count;
17
18  always @(posedge fifoif.clk or negedge fifoif.rst_n) begin
19      if (!fifoif.rst_n) begin
20          wr_ptr <= 0;
21          fifoif.overflow <=0;  // bug1
22          fifoif.wr_ack <=0;
23      end
24      else if (fifoif.wr_en && count < fifoif.FIFO_DEPTH) begin
25          mem[wr_ptr] <= fifoif.data_in;
26          fifoif.wr_ack <= 1;
27          wr_ptr <= wr_ptr + 1;
28         fifoif.overflow <=0;
29      end
30      else begin
31          fifoif.wr_ack <= 0;
32          if (fifoif.full && fifoif.wr_en)  // bug2
33              fifoif.overflow <= 1;
34          else
35              fifoif.overflow <= 0;
36      end
37  end
38
39  always @(posedge fifoif.clk or negedge fifoif.rst_n) begin
40      if (!fifoif.rst_n) begin
41          rd_ptr <= 0;
42          fifoif.underflow <=0;
43      end
44      else if (fifoif.rd_en && count != 0) begin
45          fifoif.data_out <= mem[rd_ptr];
46          rd_ptr <= rd_ptr + 1;
47          fifoif.underflow<=0;
48      end
```

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

always @(posedge fifoif.clk or negedge fifoif.rst_n) begin
    if (!fifoif.rst_n) begin
        count <= 0;
    end
    else begin
        if  ( ({fifoif.wr_en, fifoif.rd_en} == 2'b10) && !fifoif.full)
            count <= count + 1;
        else if ( ({fifoif.wr_en, fifoif.rd_en} == 2'b01) && !fifoif.empty)
            count <= count - 1;
          else if ({fifoif.wr_en , fifoif.rd_en} == 2'b11) begin   //bug3
            if (fifoif.full)
            count<=count-1;
            if (fifoif.empty)
            count<=count+1;
          end
    end
end

assign fifoif.full = (count == fifoif.FIFO_DEPTH)? 1 : 0;
assign fifoif.empty = (count == 0)? 1 : 0;
assign fifoif.almostfull = (count == fifoif.FIFO_DEPTH-1)? 1 : 0; //bug4
assign fifoif.almostempty = (count == 1)? 1 : 0;


endmodule

```

Assertions:

```
module FIFO_SVA(FIFO_if.DUT fifoif);
    property no_1;
@(posedge fifoif.clk) !fifoif.rst_n |=> $past(!FIFO.wr_ptr) && $past(!FIFO.rd_ptr) && FIFO.count == 0;
endproperty
property no_2;
@(posedge fifoif.clk) disable iff (!fifoif.rst_n) fifoif.wr_en && !fifoif.full |=> fifoif.wr_ack;
endproperty
property no_3;
@(posedge fifoif.clk) disable iff (!fifoif.rst_n) fifoif.wr_en && fifoif.full |=> fifoif.overflow;
endproperty
property no_4;
@(posedge fifoif.clk) disable iff (!fifoif.rst_n) fifoif.rd_en && fifoif.empty |=> fifoif.underflow;
endproperty
property no_5;
@(posedge fifoif.clk) disable iff (!fifoif.rst_n) FIFO.count==0 |=> $past(fifoif.empty);
endproperty
property no_6;
@(posedge fifoif.clk) disable iff (!fifoif.rst_n) FIFO.count==fifoif.FIFO_DEPTH |=> $past(fifoif.full);
endproperty
property no_7;
@(posedge fifoif.clk) disable iff (!fifoif.rst_n) FIFO.count==fifoif.FIFO_DEPTH-1 |=> $past(fifoif.almostfull);
endproperty
property no_8;
@(posedge fifoif.clk) disable iff (!fifoif.rst_n) FIFO.count==1 |=> $past(fifoif.almostempty);
endproperty
property no_9;
@(posedge fifoif.clk) disable iff (!fifoif.rst_n) fifoif.wr_en && !fifoif.full && FIFO.wr_ptr==fifoif.FIFO_DEPTH-1 |=> FIFO.wr_ptr==0 ;
endproperty
property no_10;
@(posedge fifoif.clk) disable iff (!fifoif.rst_n)  fifoif.rd_en && !fifoif.empty && FIFO.rd_ptr==fifoif.FIFO_DEPTH-1 |=>  FIFO.rd_ptr==0 ;
endproperty
property no_11;
@(posedge fifoif.clk) disable iff (!fifoif.rst_n) FIFO.wr_ptr<=fifoif.FIFO_DEPTH-1 && FIFO.rd_ptr<=fifoif.FIFO_DEPTH-1 && FIFO.count<=fifoif.FIFO_DEPTH ;
endproperty
```

```
reset_assert:assert property (no_1);
reset_cover:cover property (no_1);
wr_ack_assert:assert property (no_2);
wr_ack_cover:cover property (no_2);
overflow_assert:assert property (no_3);
overflow_cover:cover property (no_3);
underflow_assert:assert property (no_4);
underflow_cover:cover property (no_4);
empty_assert:assert property (no_5);
empty_cover:cover property (no_5);
full_assert:assert property (no_6);
full_cover:cover property (no_6);
almostfull_assert:assert property (no_7);
almostfull_cover:cover property (no_7);
almostempty_assert:assert property (no_8);
almostempty_cover:cover property (no_8);
wraparound_wr_assert:assert property (no_9);
wraparound_wr_cover:cover property (no_9);
wraparound_rd_assert:assert property (no_10);
wraparound_rd_cover:cover property (no_10);
threshold_assert:assert property (no_11);
threshold_cover:cover property (no_11);
endmodule
```

Interface:

```
interface FIFO_if (clk);
parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;

input bit clk;

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

endinterface
```

Top:

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

module FIFO_top();
    bit clk;

    initial begin
        forever
            #1 clk = ~clk;
    end

    FIFO_if fifoif(clk);
    FIFO DUT(fifoif);
    bind FIFO FIFO_SVA FIFO_SVA_inst(fifoif);

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

Test:

```
package FIFO_test_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
import FIFO_env_pkg::*;
import FIFO_reset_sequence_pkg::*;
import FIFO_write_sequence_pkg::*;
import FIFO_read_sequence_pkg::*;
import FIFO_write_read_sequence_pkg::*;
import FIFO_config_pkg::*;

class FIFO_test extends uvm_test;
    `uvm_component_utils(FIFO_test)
    virtual FIFO_if FIFO_vif;
    FIFO_env env;
    FIFO_write_sequence write_seq;
    FIFO_read_sequence read_seq;
    FIFO_write_read_sequence write_read_seq;
    FIFO_reset_sequence reset_seq;
    FIFO_config FIFO_cfg;

    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);
        env = FIFO_env::type_id::create("env",this);
        FIFO_cfg = FIFO_config::type_id::create("FIFO_cfg",this);
        write_seq = FIFO_write_sequence::type_id::create("write_seq",this);
        read_seq = FIFO_read_sequence::type_id::create("read_seq",this);
        write_read_seq = FIFO_write_read_sequence::type_id::create("write_read_seq",this);
        reset_seq = FIFO_reset_sequence::type_id::create("reset_seq",this);

        if(!uvm_config_db #(virtual FIFO_if)::get(this, "","FIFO_IF",FIFO_cfg.FIFO_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_cfg);
    endfunction
```

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

        `uvm_info("run_phase", "Reset_Assertion", UVM_LOW)
        reset_seq.start(env.agt.sqr);

        `uvm_info("run_phase", "Reset_Deassertion", UVM_LOW)
        `uvm_info("run_phase", "Stimulus Generation Started_1", UVM_LOW)
        write_seq.start(env.agt.sqr);
        `uvm_info("run_phase", "Stimulus Generation Ended_1", UVM_LOW)

        `uvm_info("run_phase", "Stimulus Generation Started_2", UVM_LOW)
        read_seq.start(env.agt.sqr);
        `uvm_info("run_phase", "Stimulus Generation Ended_2", UVM_LOW)

        `uvm_info("run_phase", "Stimulus Generation Started_3", UVM_LOW)
        write_read_seq.start(env.agt.sqr);
        `uvm_info("run_phase", "Stimulus Generation Ended_3", UVM_LOW)

        `uvm_info("run_phase", "Reset_Assertion", UVM_LOW)
        reset_seq.start(env.agt.sqr);
        `uvm_info("run_phase", "Reset_Deassertion", UVM_LOW)

        phase.drop_objection(this);
    endtask
endclass
endpackage
```

Environment:

```
package FIFO_env_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
import FIFO_agent_pkg::*;
import FIFO_scoreboard_pkg::*;
import FIFO_coverage_pkg::*;

class FIFO_env extends uvm_env;
    `uvm_component_utils(FIFO_env)
    FIFO_agent agt;
    FIFO_scoreboard sb;
    FIFO_coverage cov;

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

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

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

Agent:

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

class FIFO_agent extends uvm_agent;
    `uvm_component_utils(FIFO_agent)
    FIFO_sequencer sqr;
    FIFO_driver drv;
    FIFO_monitor mon;
    FIFO_config FIFO_cfg;
    uvm_analysis_port #(FIFO_sequence_item) agt_ap;

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

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        if(!uvm_config_db #(FIFO_config)::get(this,"","CFG",FIFO_cfg)) begin
            `uvm_fatal("build_phase", "Test - Unable to get configuration object")
        end
        sqr = FIFO_sequencer::type_id::create("sqr",this);
        drv = FIFO_driver::type_id::create("drv",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);
        drv.FIFO_vif = FIFO_cfg.FIFO_vif;
        mon.FIFO_vif = FIFO_cfg.FIFO_vif;
        drv.seq_item_port.connect(sqr.seq_item_export);
        mon.mon_ap.connect(agt_ap);
    endfunction
endclass
endpackage
```

Config:

```
1   package FIFO_config_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4
5   class FIFO_config extends uvm_object;
6       `uvm_object_utils(FIFO_config)
7       virtual FIFO_if FIFO_vif;
8
9       function new(string name = "FIFO_config");
10          super.new(name);
11      endfunction
12  endclass
13  endpackage
```

Sequence item:

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

class FIFO_sequence_item extends uvm_sequence_item;
    `uvm_object_utils(FIFO_sequence_item)

parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;

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

integer RD_EN_ON_DIST=30,WR_EN_ON_DIST=70;

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

constraint rst_n_const {rst_n dist {0:=2,1:=98};}
constraint wr_en_const {
    wr_en dist{1:=WR_EN_ON_DIST,0:=(100-WR_EN_ON_DIST)};
}
constraint rd_en_const {
    rd_en dist{1:=RD_EN_ON_DIST,0:=(100-RD_EN_ON_DIST)};
}
    function string convert2string();
        return $sformatf("%s rst_n = 0x%0b, wr_en = 0x%0b, rd_en = 0x%0b, data_in = %0h ,data_out = %0h,
        wr_ack=%0b,full=%0b,empty=%0b,overflow=%0b,underflow=%0b,
        almostfull=%0b,almostempty=%0b",super.convert2string(),rst_n,wr_en,rd_en,data_in,data_out,wr_ack,full,
        empty,overflow,underflow,almostfull,almostempty);
    endfunction

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


endclass
endpackage
```

Reset sequence:

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

class FIFO_reset_sequence extends uvm_sequence #(FIFO_sequence_item);
    `uvm_object_utils(FIFO_reset_sequence)
    FIFO_sequence_item seq_item;

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

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

Write sequence:

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

class FIFO_write_sequence extends uvm_sequence #(FIFO_sequence_item);
    `uvm_object_utils(FIFO_write_sequence)
    FIFO_sequence_item seq_item;

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

    task body;
        seq_item = FIFO_sequence_item::type_id::create("seq_item");
        repeat(1000) begin
            start_item(seq_item);
            seq_item.randomize(rst_n);
            seq_item.wr_en=1;
            seq_item.rd_en=0;
            seq_item.randomize(data_in);
            finish_item(seq_item);
        end
    endtask
endclass
endpackage
```

Read sequence:

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

class FIFO_read_sequence extends uvm_sequence #(FIFO_sequence_item);
    `uvm_object_utils(FIFO_read_sequence)
    FIFO_sequence_item seq_item;

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

    task body;
        seq_item = FIFO_sequence_item::type_id::create("seq_item");
        repeat(1000) begin
            start_item(seq_item);
            seq_item.randomize(rst_n);
            seq_item.wr_en=0;
            seq_item.rd_en=1;
            seq_item.randomize(data_in);
            finish_item(seq_item);
        end
    endtask
endclass
endpackage
```

Write and read sequence:

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

class FIFO_write_read_sequence extends uvm_sequence #(FIFO_sequence_item);
    `uvm_object_utils(FIFO_write_read_sequence)
    FIFO_sequence_item seq_item;

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

    task body;
        seq_item = FIFO_sequence_item::type_id::create("seq_item");
        repeat(1000) begin
            start_item(seq_item);
            seq_item.randomize(rst_n);
            seq_item.wr_en=1;
            seq_item.rd_en=1;
            seq_item.randomize(data_in);
            finish_item(seq_item);
        end
    endtask
endclass
endpackage
```

Sequencer:

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

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

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

Driver:

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

class FIFO_driver extends uvm_driver #(FIFO_sequence_item);
    `uvm_component_utils(FIFO_driver)
    virtual FIFO_if FIFO_vif;
    FIFO_sequence_item stim_seq_item;

    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
            stim_seq_item = FIFO_sequence_item::type_id::create("stim_seq_item");
            seq_item_port.get_next_item(stim_seq_item);

            FIFO_vif.data_in = stim_seq_item.data_in;
            FIFO_vif.rst_n = stim_seq_item.rst_n;
            FIFO_vif.wr_en = stim_seq_item.wr_en;
            FIFO_vif.rd_en = stim_seq_item.rd_en;

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

Monitor:

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

class FIFO_monitor extends uvm_monitor;
    `uvm_component_utils(FIFO_monitor)
    virtual FIFO_if FIFO_vif;
    FIFO_sequence_item rsp_seq_item;
    uvm_analysis_port #(FIFO_sequence_item) mon_ap;

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

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

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever begin
            rsp_seq_item = FIFO_sequence_item::type_id::create("rsp_seq_item");
            @(negedge FIFO_vif.clk);

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

            mon_ap.write(rsp_seq_item);
            `uvm_info("run_phase", rsp_seq_item.convert2string(), UVM_HIGH)
        end
    endtask
endclass
endpackage
```

Coverage:

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

class FIFO_coverage extends uvm_component;
    `uvm_component_utils(FIFO_coverage)
    uvm_analysis_export #(FIFO_sequence_item) cov_export;
    uvm_tlm_analysis_fifo #(FIFO_sequence_item) cov_fifo;
    FIFO_sequence_item seq_item_cov;

    covergroup cg ;
   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;
   overflow_cp:coverpoint seq_item_cov.overflow;
   underflow_cp:coverpoint seq_item_cov.underflow;
   full_cp:coverpoint seq_item_cov.full;
   almostempty_cp:coverpoint seq_item_cov.almostempty;
   almostfull_cp:coverpoint seq_item_cov.almostfull;
   empty_cp:coverpoint seq_item_cov.empty;

   cross_1: cross wr_en_cp,rd_en_cp,wr_ack_cp;
   cross_2: cross wr_en_cp,rd_en_cp,overflow_cp;
   cross_3: cross wr_en_cp,rd_en_cp,underflow_cp;
   cross_4: cross wr_en_cp,rd_en_cp,full_cp;
   cross_5: cross wr_en_cp,rd_en_cp,almostempty_cp;
   cross_6: cross wr_en_cp,rd_en_cp,almostfull_cp;
   cross_7: cross wr_en_cp,rd_en_cp,empty_cp;
    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
```

```
43      function void connect_phase(uvm_phase phase);
44          super.connect_phase(phase);
45          cov_export.connect(cov_fifo.analysis_export);
46      endfunction
47
48      task run_phase(uvm_phase phase);
49          super.run_phase(phase);
50          forever begin
51              cov_fifo.get(seq_item_cov);
52              cg.sample();
53          end
54      endtask
55  endclass
56  endpackage
```

Scoreboard:

```
1   package FIFO_scoreboard_pkg;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   import FIFO_sequence_item_pkg::*;
5
6   class FIFO_scoreboard extends uvm_scoreboard;
7       `uvm_component_utils(FIFO_scoreboard)
8       uvm_analysis_export #(FIFO_sequence_item) sb_export;
9       uvm_tlm_analysis_fifo #(FIFO_sequence_item) sb_fifo;
10      FIFO_sequence_item seq_item_sb;
11
12      int error_count = 0;
13      int correct_count = 0;
14
15  parameter FIFO_WIDTH = 16;
16  parameter FIFO_DEPTH = 8;
17
18  logic [FIFO_WIDTH-1:0] data_out_ref;
19  logic [FIFO_WIDTH-1:0] fifo_queue [$];
20  int count_fifo =0;
21
22     function new(string name = "FIFO_scoreboard", uvm_component parent = null);
23         super.new(name,parent);
24      endfunction
25
26      function void build_phase(uvm_phase phase);
27          super.build_phase(phase);
28          sb_export = new("sb_export",this);
29          sb_fifo = new("sb_fifo",this);
30      endfunction
31
32      function void connect_phase(uvm_phase phase);
33          super.connect_phase(phase);
34          sb_export.connect(sb_fifo.analysis_export);
35      endfunction
36
```

```
    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, transaction received by the DUT: %s while the reference out: %0d",seq_item_sb.convert2string(),data_out_ref));
                error_count++;
            end
            else begin
                `uvm_info("run_phase", $sformatf("correct FIFO out: %s",seq_item_sb.convert2string()), UVM_HIGH)
                correct_count++;
            end
        end
    endtask

function void reference_model(input FIFO_sequence_item seq_item_sb);
if(!seq_item_sb.rst_n) begin
    fifo_queue <= {};
    count_fifo <= 0 ;
end
else begin
    if (seq_item_sb.wr_en && count_fifo < FIFO_DEPTH) begin
        fifo_queue.push_back(seq_item_sb.data_in);
        count_fifo <= fifo_queue.size();
    end
    if (seq_item_sb.rd_en && count_fifo!=0) begin
        data_out_ref <= fifo_queue.pop_front();
        count_fifo <= fifo_queue.size();
    end
end
endfunction

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

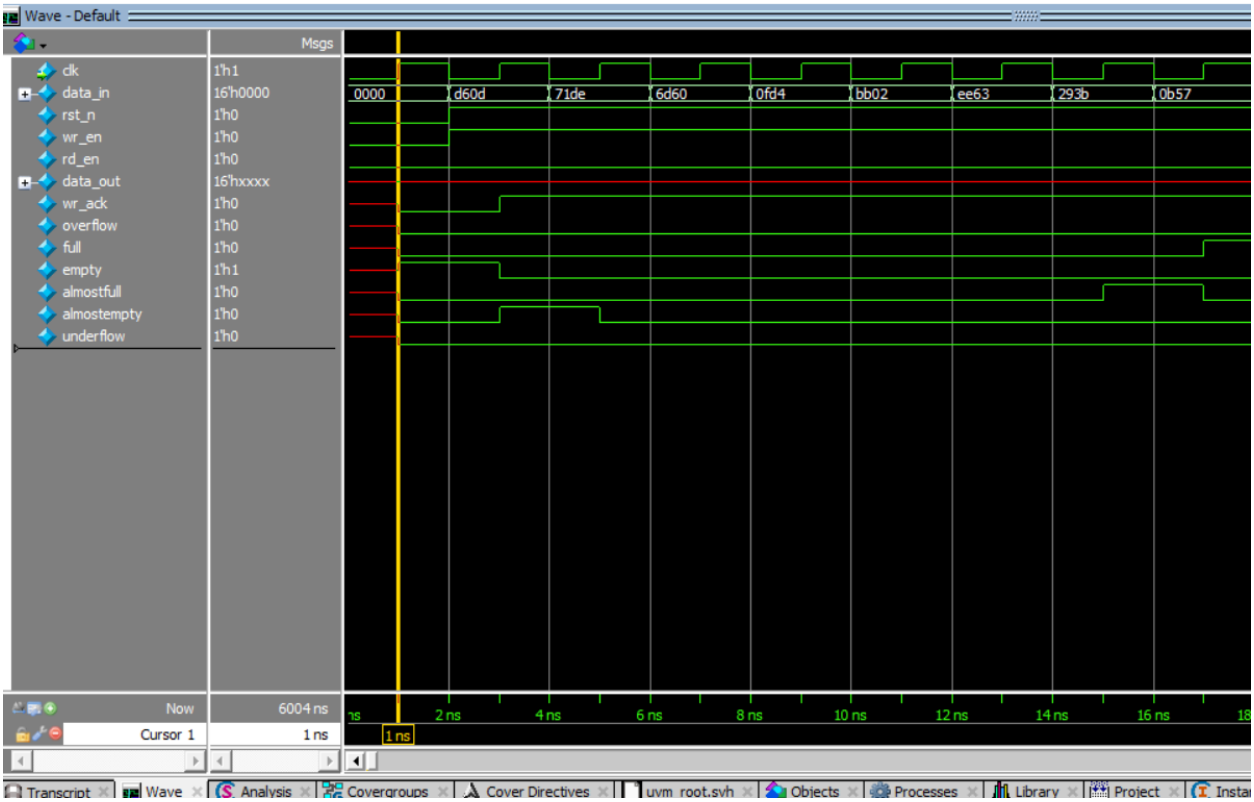
Do file:

```
vlib work
vlog -f src_files_FIFO.list +cover -covercells
vsim -voptargs=+acc work.FIFO_top -cover -classdebug -uvmcontrol=all
add wave /FIFO_top/fifoif/*
coverage save FIFO_tb.ucdb -onexit -du FIFO
run -all
```

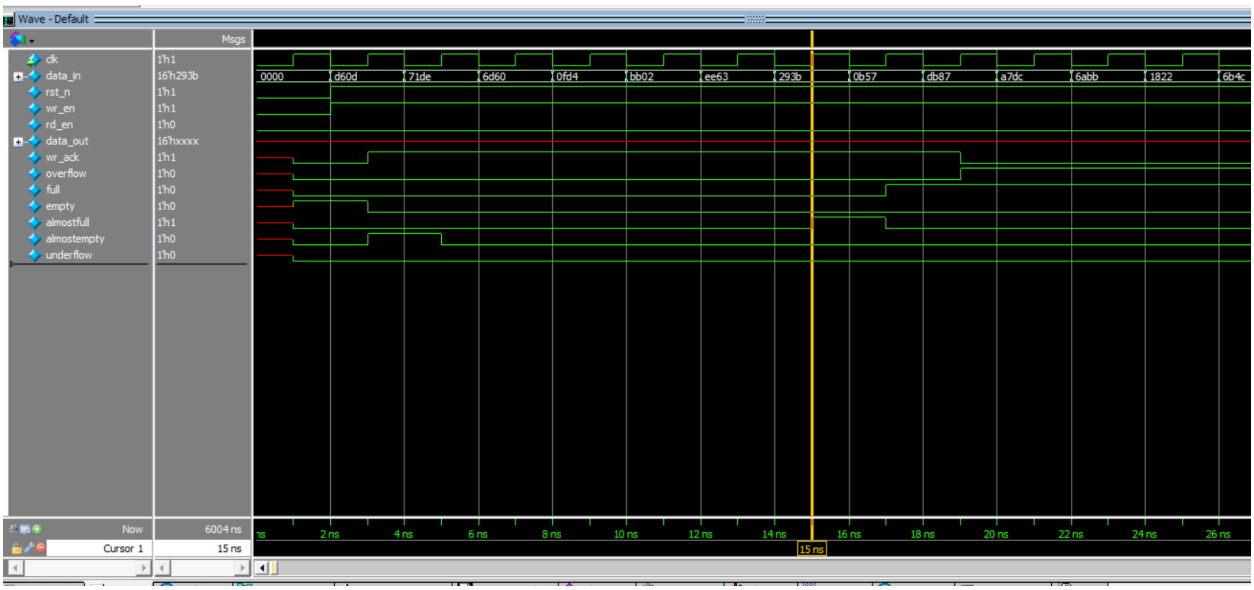
Src_files_FIFO.list :

```
FIFO_if.sv
FIFO.sv
FIFO_SVA.sv
FIFO_config.sv
FIFO_sequence_item.sv
FIFO_reset_sequence.sv
FIFO_write_sequence.sv
FIFO_read_sequence.sv
FIFO_write_read_sequence.sv
FIFO_sequencer.sv
FIFO_monitor.sv
FIFO_driver.sv
FIFO_scoreboard.sv
FIFO_coverage.sv
FIFO_agent.sv
FIFO_env.sv
FIFO_test.sv
FIFO_top.sv
```

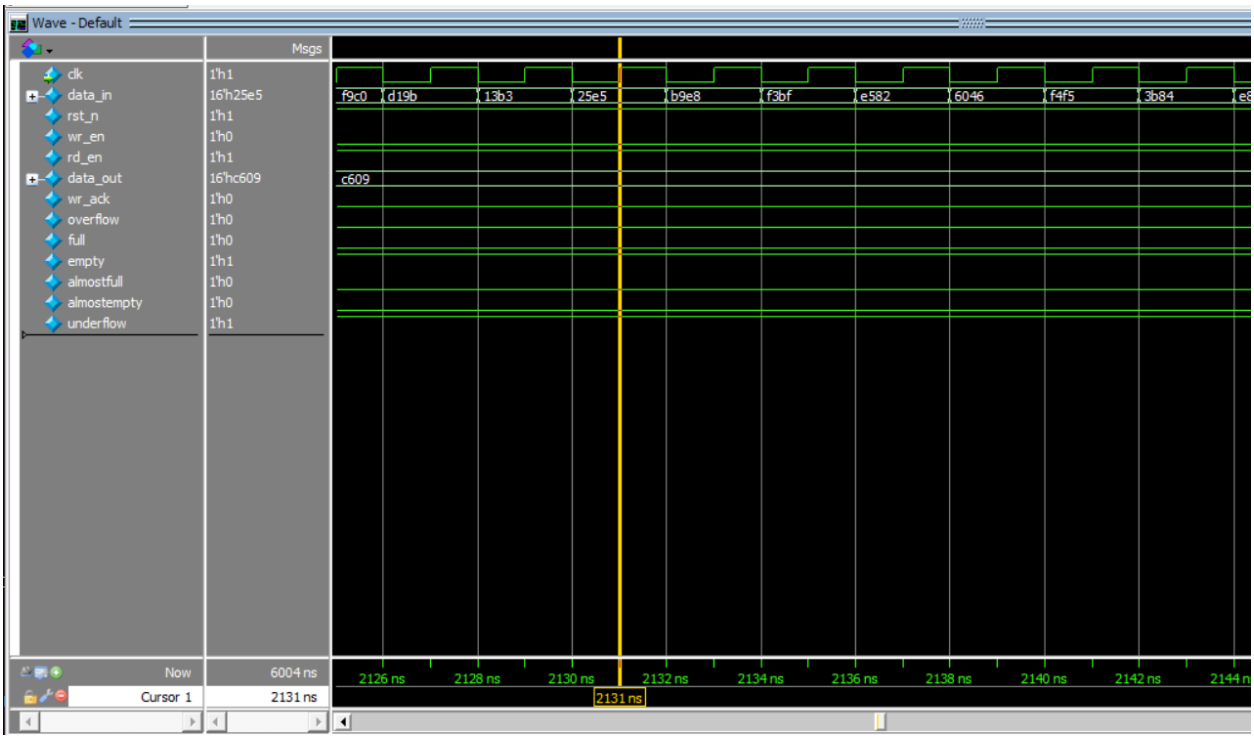
Reset sequence wave:



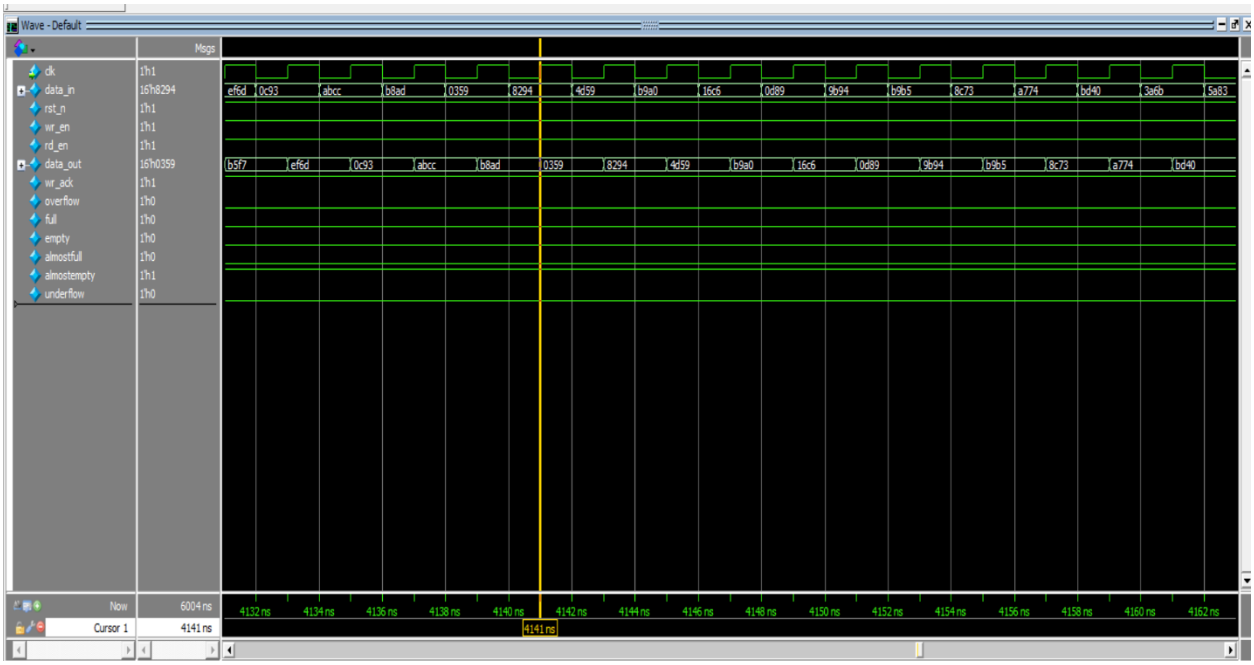
Write sequence wave:



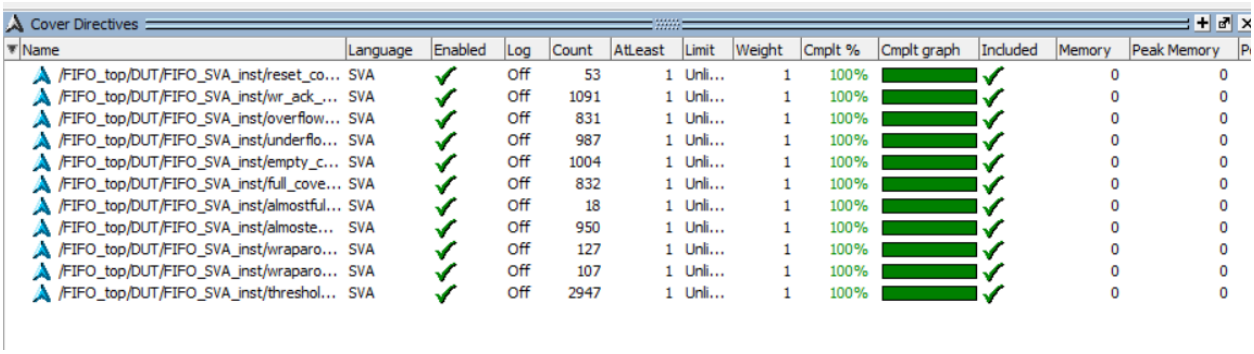
Read sequence wave:



Write and read sequence wave:



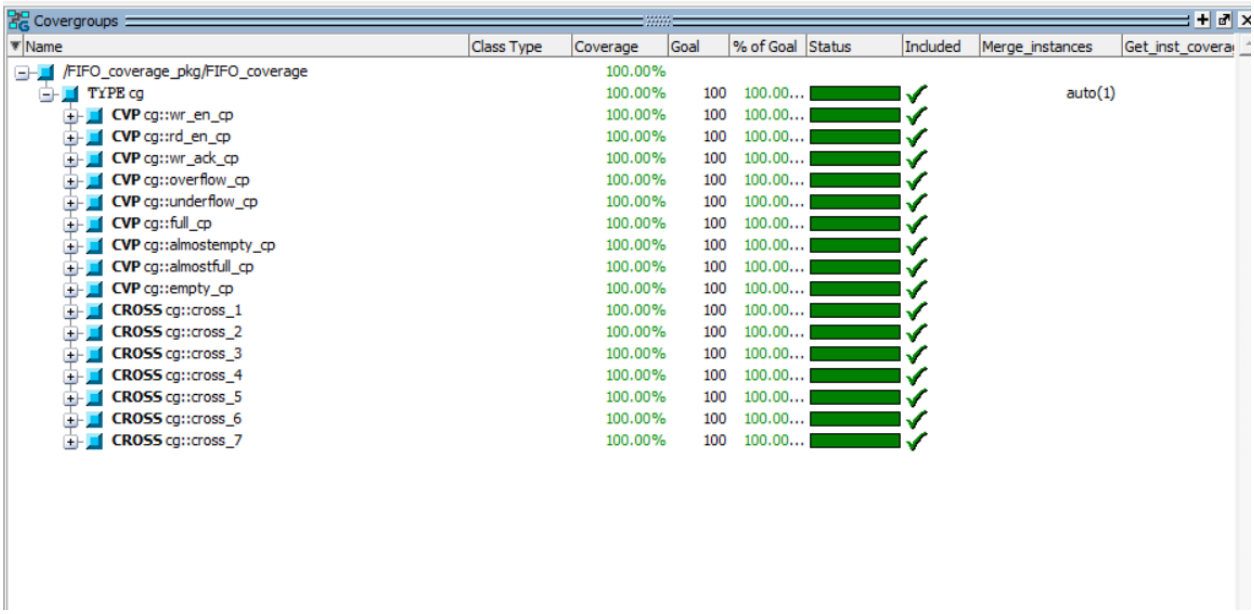
Assertion coverage:



Cover Directives

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory
/FIFO_top/DUT/FIFO_SVA_inst/reset_co...	SVA	✓	Off	53	1	Unli...	1	100%	✓	✓	0	0
/FIFO_top/DUT/FIFO_SVA_inst/wr_ack_...	SVA	✓	Off	1091	1	Unli...	1	100%	✓	✓	0	0
/FIFO_top/DUT/FIFO_SVA_inst/overflow...	SVA	✓	Off	831	1	Unli...	1	100%	✓	✓	0	0
/FIFO_top/DUT/FIFO_SVA_inst/underflo...	SVA	✓	Off	987	1	Unli...	1	100%	✓	✓	0	0
/FIFO_top/DUT/FIFO_SVA_inst/empty_c...	SVA	✓	Off	1004	1	Unli...	1	100%	✓	✓	0	0
/FIFO_top/DUT/FIFO_SVA_inst/full_cove...	SVA	✓	Off	832	1	Unli...	1	100%	✓	✓	0	0
/FIFO_top/DUT/FIFO_SVA_inst/almostful...	SVA	✓	Off	18	1	Unli...	1	100%	✓	✓	0	0
/FIFO_top/DUT/FIFO_SVA_inst/almoste...	SVA	✓	Off	950	1	Unli...	1	100%	✓	✓	0	0
/FIFO_top/DUT/FIFO_SVA_inst/wraparo...	SVA	✓	Off	127	1	Unli...	1	100%	✓	✓	0	0
/FIFO_top/DUT/FIFO_SVA_inst/wraparo...	SVA	✓	Off	107	1	Unli...	1	100%	✓	✓	0	0
/FIFO_top/DUT/FIFO_SVA_inst/threshol...	SVA	✓	Off	2947	1	Unli...	1	100%	✓	✓	0	0

Functional coverage:



Covergroups

Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_covera
/FIFO_coverage_pkg/FIFO_coverage		100.00%						
TYPE cg		100.00%	100	100.00...	✓	✓	auto(1)	
CVP cg::wr_en_cp		100.00%	100	100.00...	✓	✓		
CVP cg::rd_en_cp		100.00%	100	100.00...	✓	✓		
CVP cg::wr_ack_cp		100.00%	100	100.00...	✓	✓		
CVP cg::overflow_cp		100.00%	100	100.00...	✓	✓		
CVP cg::underflow_cp		100.00%	100	100.00...	✓	✓		
CVP cg::full_cp		100.00%	100	100.00...	✓	✓		
CVP cg::almostempty_cp		100.00%	100	100.00...	✓	✓		
CVP cg::almostfull_cp		100.00%	100	100.00...	✓	✓		
CVP cg::empty_cp		100.00%	100	100.00...	✓	✓		
CROSS cg::cross_1		100.00%	100	100.00...	✓	✓		
CROSS cg::cross_2		100.00%	100	100.00...	✓	✓		
CROSS cg::cross_3		100.00%	100	100.00...	✓	✓		
CROSS cg::cross_4		100.00%	100	100.00...	✓	✓		
CROSS cg::cross_5		100.00%	100	100.00...	✓	✓		
CROSS cg::cross_6		100.00%	100	100.00...	✓	✓		
CROSS cg::cross_7		100.00%	100	100.00...	✓	✓		

Code coverage:

```
# ================================================================================
# Branch Coverage:
#     Enabled Coverage              Bins      Hits    Misses  Coverage
#     ----------------              ----      ----    ------  --------
#     Branches                        26        26         0   100.00%
#
# ================================Branch Details================================
#
# Branch Coverage for instance /\FIFO_top#DUT
#
#     Line         Item                      Count     Source
#     ----         ----                      -----     ------
#   File FIFO.sv
# ------------------------------------IF Branch------------------------------------
#     19                                     3055     Count coming in to IF
#     19              1                       108      if (!fifoif.rst_n) begin
#
#     24              1                       1114     else if (fifoif.wr_en && count < fifoif.FIFO_DEPTH) begin
#
#     30              1                       1833     else begin
#
# Branch totals: 3 hits of 3 branches = 100.00%
#
# ------------------------------------IF Branch------------------------------------
#     32                                     1833     Count coming in to IF
#     32              1                       847       if (fifoif.full && fifoif.wr_en)  // bug2
#
#     34              1                       986       else
#
# Branch totals: 2 hits of 2 branches = 100.00%
#
# ------------------------------------IF Branch------------------------------------
#     40                                     3055     Count coming in to IF
#     40              1                       108      if (!fifoif.rst_n) begin
#
#     44              1                       963      else if (fifoif.rd_en && count != 0) begin
```

```
#
# Statement Coverage:
#     Enabled Coverage              Bins      Hits    Misses  Coverage
#     ----------------              ----      ----    ------  --------
#     Statements                      28        28         0   100.00%
#
# ================================Statement Details================================
#
# Statement Coverage for instance /\FIFO_top#DUT  --
#
#     Line         Item                      Count     Source
#     ----         ----                      -----     ------
#   File FIFO.sv
#     8                                                module FIFO(FIFO_if.DUT fifoif);
#
#     9
#
#     10
#
#     11                                               localparam max_fifo_addr = $clog2(fifoif.FIFO_DEPTH);
#
#     12
#
#     13                                               reg [fifoif.FIFO_WIDTH-1:0] mem [fifoif.FIFO_DEPTH-1:0];
#
#     14
#
#     15                                               reg [max_fifo_addr-1:0] wr_ptr, rd_ptr;
#
#     16                                               reg [max_fifo_addr:0] count;
#
#     17
#
#     18              1                       3055     always @(posedge fifoif.clk or negedge fifoif.rst_n) begin
#
#     19                                                if (!fifoif.rst_n) begin
#
#     20              1                        108       wr_ptr <= 0;
```

```
#
# Toggle Coverage:
#     Enabled Coverage              Bins      Hits    Misses  Coverage
#     ----------------              ----      ----    ------  --------
#     Toggles                         20        20         0   100.00%
#
# ================================Toggle Details================================
#
# Toggle Coverage for instance /\FIFO_top#DUT  --
#
#                                             Node      1H->0L      0L->1H  "Coverage"
#                                             ----------------------------------------
#                                       count[0-3]           1           1      100.00
#                                      rd_ptr[0-2]           1           1      100.00
#                                      wr_ptr[0-2]           1           1      100.00
#
# Total Node Count     =         10
# Toggled Node Count   =         10
# Untoggled Node Count =          0
#
# Toggle Coverage      =     100.00% (20 of 20 bins)
#
#
# DIRECTIVE COVERAGE:
# ---------------------------------------------------------------------------------------------------
# Name                                     Design Design   Lang File(Line)      Hits Status
#                                          Unit   UnitType
# ---------------------------------------------------------------------------------------------------
# /\FIFO_top#DUT /FIFO_SVA_inst/reset_cover
#                                          FIFO_SVA Verilog  SVA  FIFO_SVA.sv(38)   53 Covered
# /\FIFO_top#DUT /FIFO_SVA_inst/wr_ack_cover
#                                          FIFO_SVA Verilog  SVA  FIFO_SVA.sv(40) 1091 Covered
# /\FIFO_top#DUT /FIFO_SVA_inst/overflow_cover
#                                          FIFO_SVA Verilog  SVA  FIFO_SVA.sv(42)  831 Covered
# /\FIFO_top#DUT /FIFO_SVA_inst/underflow_cover
#                                          FIFO_SVA Verilog  SVA  FIFO_SVA.sv(44)  987 Covered
# /\FIFO_top#DUT /FIFO_SVA_inst/empty_cover
#                                          FIFO_SVA Verilog  SVA  FIFO_SVA.sv(46) 1004 Covered
# /\FIFO_top#DUT /FIFO_SVA_inst/full_cover FIFO_SVA Verilog  SVA  FIFO_SVA.sv(48)  832 Covered
```

Transcript:

```
UVM_INFO @ 0: reporter [RNTST] Running test FIFO_test...
UVM_INFO FIFO_test.sv(44) @ 0: uvm_test_top [run_phase] Reset_Assertion
****************************************************************************
* 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(47) @ 2: uvm_test_top [run_phase] Reset_Deassertion
UVM_INFO FIFO_test.sv(48) @ 2: uvm_test_top [run_phase] Stimulus Generation Started_1
** Error: Assertion error.
   Time: 3 ns Started: 1 ns  Scope: FIFO_top.DUT.FIFO_SVA_inst.reset_assert File: FIFO_SVA.sv Line: 37
UVM_INFO FIFO_test.sv(50) @ 2002: uvm_test_top [run_phase] Stimulus Generation Ended_1
UVM_INFO FIFO_test.sv(52) @ 2002: uvm_test_top [run_phase] Stimulus Generation Started_2
UVM_INFO FIFO_test.sv(54) @ 4002: uvm_test_top [run_phase] Stimulus Generation Ended_2
UVM_INFO FIFO_test.sv(56) @ 4002: uvm_test_top [run_phase] Stimulus Generation Started_3
UVM_INFO FIFO_test.sv(58) @ 6002: uvm_test_top [run_phase] Stimulus Generation Ended_3
UVM_INFO FIFO_test.sv(60) @ 6002: uvm_test_top [run_phase] Reset_Assertion
UVM_INFO FIFO_test.sv(62) @ 6004: uvm_test_top [run_phase] Reset_Deassertion
UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 6004: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
UVM_INFO FIFO_scoreboard.sv(74) @ 6004: uvm_test_top.env.sb [report_phase] total successful transaction: 3001
UVM_INFO FIFO_scoreboard.sv(75) @ 6004: uvm_test_top.env.sb [report_phase] total failed transaction: 0

--- UVM Report Summary ---

** Report counts by severity
UVM_INFO :   16
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]    10
** Note: $finish    : C:/questasim64_2024.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
   Time: 6004 ns  Iteration: 61  Instance: /FIFO_top
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
```