UVM Synchronous FIFO

Submitted to: -

Eng/Kareem Waseem

Eng/Alaa

Submitted By: -

Mahmoud Lotfy AbdelHamid Amr

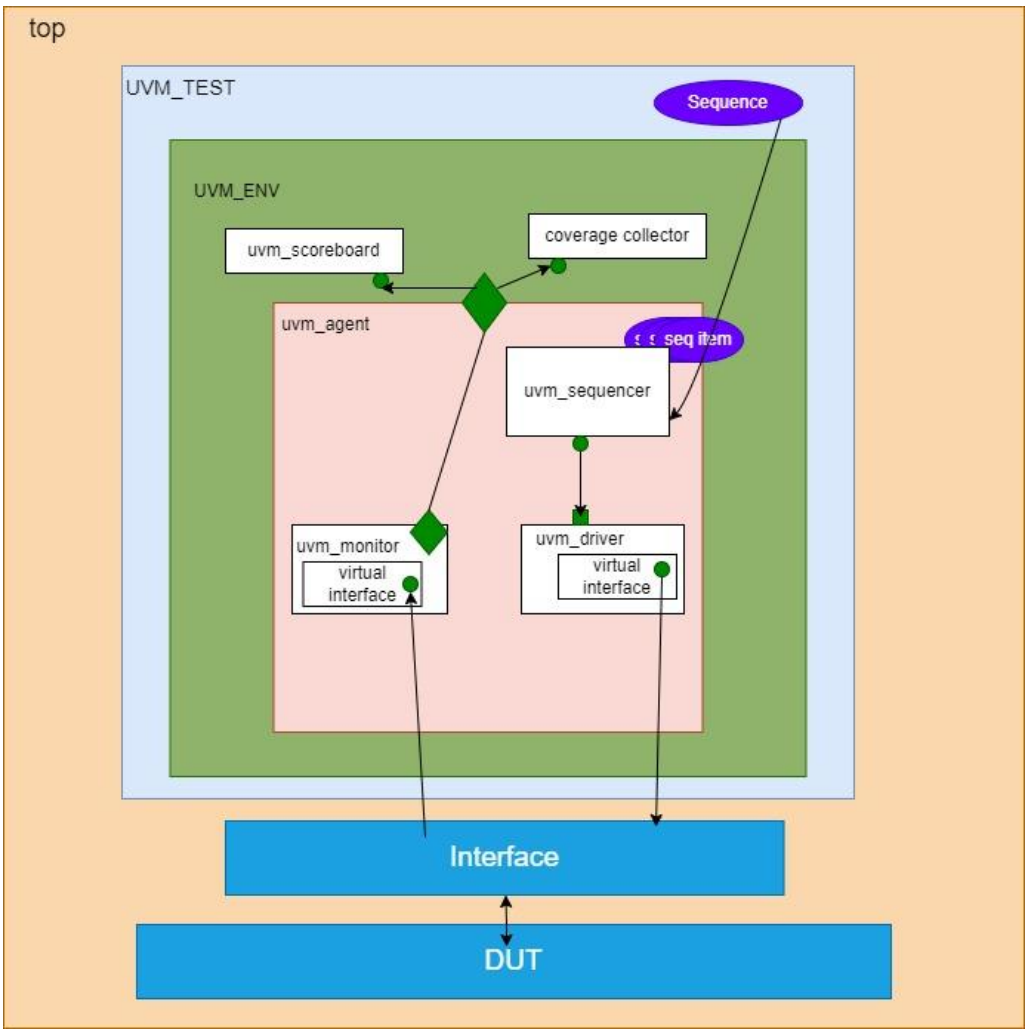
mahmoudlotfy383@gmail.com

Table of Contents

1- Verification Plan

Label	Description	Stimulus Generation	Functional Coverage	Functionality Check
RESET	When the reset is asserted, the output flags value should be low and empty flag must be high and make resrt be deasserted most of time	At the start of fifo_test then make it randomized	cover reset	immedtiate assertions to chech async reset and also check the effect of rst in score_board
FIFO_1	write sequence only	randomized at the begin of the fifo_test after rst	cover all casses of wr_en and rd_en by cross coverage in coverage collector	concurrent assertions to check all effects of wr_en and rd_en and also by score_board
FIFO_2	read sequence only	randomized after write sequence only	cover all casses of wr_en and rd_en by cross coverage in coverage collector	concurrent assertions to check all effects of wr_en and rd_en and also by score_board
FIFO_3	write read sequence only	randomized after read sequence only	cover all casses of wr_en and rd_en by cross coverage in coverage collector	concurrent assertions to check all effects of wr_en and rd_en and also by score_board
FIFO_4	the effect when wr_en high and rd_en low should write	randomized during the fifo_test	cover all casses of wr_en and rd_en by cross coverage in coverage collector	concurrent assertions to check all effects of wr_en and rd_en and also by score_board
FIFO_5	the effect when wr_en low and rd_en high should read	randomized during the fifo_test	cover all casses of wr_en and rd_en by cross coverage in coverage collector	concurrent assertions to check all effects of wr_en and rd_en and also by score_board
FIFO_6	the effect when wr_en high and rd_en high and full should rea	randomized during the fifo_test	cover all casses of wr_en and rd_en by cross coverage in coverage collector	concurrent assertions to check all effects of wr_en and rd_en and also by score_board
FIFO_7	the effect when wr_en high and rd_en high and empty write	randomized during the fifo_test	cover all casses of wr_en and rd_en by cross coverage in coverage collector	concurrent assertions to check all effects of wr_en and rd_en and also by score_board
FIFO_8	wr_en high and full	randomized during the fifo_test	cover all casses of wr_en and rd_en and full by cross coverage in coverage collector	concurrent assertions to check all effects of wr_en and also by score_board
FIFO_9	rd_en high and empty	randomized during the fifo_test	cover all casses of wr_en and rd_en and empty by cross coverage in coverage collector	concurrent assertions to check all effects of rd_en and also by score_board
FIFO_10	check almostfull	randomized during the fifo_test	cover all casses of wr_en and rd_en and almostfull by cross coverage	concurrent assertions to check almostfull and also by score_board
FIFO_11	check almostempty	randomized during the fifo_test	cover all casses of wr_en and rd_en and almostempty by cross coverage in coverage collector	concurrent assertions to check almostempty and also by score_board
FIFO_12	check overflow	randomized during the fifo_test	cover all casses of wr_en and rd_en and overflow by cross coverage in coverage collector	concurrent assertions to check overflow and also by score_board
FIFO_13	check underflow	randomized during the fifo_test	cover all casses of wr_en and rd_en and underflow by cross coverage in coverage collector	concurrent assertions to check underflow and also by score_board
FIFO_14	check relation between full and almostfull	randomized during the fifo_test	_____	concurrent assertions to check relation between full and almostfull
FIFO_15	check relation between empty and almostempty	randomized during the fifo_test	_____	concurrent assertions to check relation between empty and almostempty
FIFO_16	check wr_ptr	randomized during the fifo_test	_____	concurrent assertions to check all possible casses for wr_ptr
FIFO_17	check rd_ptr	randomized during the fifo_test	_____	concurrent assertions to check all possible casses for rd_ptr
FIFO_18	check count	randomized during the fifo_test	_____	concurrent assertions to check all possible casses for count
FIFO_19	check max_fifo_addr	randomized during the fifo_test	_____	immediate assertions to check max_fifo_addr

2- UVM Structure



TLM Port

TLM export

Analysis port

Component

Object

Description: -

Top: -

generates a clock, instantiates the FIFO and its interface, binds assertions to the DUT, configures the virtual interface for UVM, and runs the UVM test.

Uvm_test: -

The test class (FIFO_test_class) is responsible for setting up the environment, configuring the FIFO, and running the test sequences.
In build phase: - The environment (env), configuration (FIFO_cnfg), and sequences (reset_seq, write_only_seq, read_only_seq, and write_read_seq) are created.
The virtual interface (FIFO_if) is retrieved and configured.
In run phase: -The test performs a reset, followed by write, read, and combined write-read operations on the FIFO using the respective sequences.

uvm_env: -

The FIFO_env_class is responsible for setting up the UVM environment for the FIFO testbench. It manages the following components:
Agent (agt): Responsible for generating stimuli and monitoring the DUT.
Scoreboard (sb): Compares actual DUT output to the expected output.
Coverage (cov): Tracks functional coverage to ensure all test scenarios are covered.
In Build Phase: Instantiates the agent, scoreboard, and coverage components.
Connect Phase: Connects the agent's analysis port to the scoreboard and coverage components

Sequence: -

Have multiple sequences
FIFO_sequence_reset: Resets the FIFO by driving the reset signal (rst_n).
write_only_sequence: Drives only write transactions to the FIFO.
read_only_sequence: Drives only read transactions to the FIFO.
write_read_sequence: Randomly drives both write and read operations.

Uvm_scoreboard: -

The FIFO_score class serves as a scoreboard that compares the outputs of the FIFO DUT with expected reference values to ensure the outputs are valid .

Coverage_Collector: -

The FIFO_coverage class serves as a coverage collector that monitors and tracks various operational states of the FIFO DUT, ensuring that all functional scenarios are being exercised during simulation.

Build Phase:
The build_phase method is responsible for initializing components within the coverage class.
cov_export: An analysis export port that connects to the DUT.
cov_fifo: An analysis FIFO for receiving sequence items for coverage analysis.
Connect Phase:
The connect_phase method establishes connections between the coverage component and other components in the UVM environment.It connects cov_export to the analysis export of cov_fifo, enabling the flow of sequence items from the DUT to the coverage collector.

Uvm_agent: -

The FIFO_agent class encapsulates the interaction between various components needed to test a FIFO design in a UVM environment.
Through the build_phase, it initializes and sets up these components, while the connect_phase ensures they are properly connected for seamless operation during simulation.

Seq_item: -

Has constraints and convert2string and convert2string_stimulus functions

Uvm_sequencer: -

The FIFO_sequencer class is responsible for Generating and sending FIFO sequence items to the FIFO driver and managing the sequencing of read and write operations to ensure correct interactions with the FIFO design.

Uvm_monitor: -

Has virtual interface
build_phase:
Creates the analysis port mon_ap during the build phase.
Run_phase:
In this task, the monitor continuously samples the signals from the FIFO DUT at each negative clock edge.
It populates the rsp_seq_item with the current values of the DUT's signals and writes it to the analysis port.
It also logs the captured sequence item for informational purposes.

Uvm_driver: -

Run_phase:

This task runs continuously to drive the DUT with stimulus.

It creates a new instance of stim_seq_item and retrieves the next sequence item from the sequencer using seq_item_port.get_next_item(stim_seq_item).

The driver sets the DUT's input signals (data_in, rst_n, wr_en, rd_en) based on the values from the stim_seq_item.

The driver waits for a negative clock edge of the DUT before signaling that the sequence item is complete with seq_item_port.item_done().

It logs the stimulus data for informational purposes using uvm_info.

3- Design &Bug Report

The design has more than bug there is the original 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
```

BUGS: -

1- Wr_ack must be rested
2- Overflow must be rested
3- almostfull must be (count == FIFO_IF.FIFO_DEPTH-1)? 1 : 0 ;
   not (count == FIFO_IF.FIFO_DEPTH-2)? 1 : 0;
4- underflow must be sequential output as required not combinational output
5- underflow must be zero if (FIFO_IF.rd_en && count != 0) and that ignored in the original design and that important to do that because not make underflow remain has old value and update each clk cycle
6- overflow must be zero if (FIFO_IF.wr_en && count < FIFO_IF.FIFO_DEPTH) and that ignored in the original design and that important to do that because not make overflow remain has old value and update each clk cycle
7- these are two conditions are ignored in design and I add them as required and the two conditions are

> Note: If a read and write enables were high and the FIFO was empty, only writing will take place and vice verse if the FIFO was full.

Must make count can deal with these two conditions

And this is the modified version of design after fixing all bugs mentioned above.

```
module FIFO(FIFO_Interface.DUT FIFO_IF);

localparam max_fifo_addr = FIFO_IF.max_fifo_addr;

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

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

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

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

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

assign FIFO_IF.full = (count == FIFO_IF.FIFO_DEPTH)? 1 : 0;
assign FIFO_IF.empty = (count == 0)? 1 : 0;
assign FIFO_IF.almostfull = (count == FIFO_IF.FIFO_DEPTH-1)? 1 : 0; /// modified (-2 => -1)
assign FIFO_IF.almostempty = (count == 1)? 1 : 0;

endmodule

```

4- Top

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

module top();

bit clk;
initial begin
    clk=0;
    forever
    #1 clk=~clk;
end
    FIFO_Interface FIFO_IF (clk);
    FIFO           DUT (FIFO_IF);
    bind           FIFO Assertions Assertions_inst (FIFO_IF);


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

5- Interface

```
interface FIFO_Interface(clk);
parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;
parameter max_fifo_addr = $clog2(FIFO_DEPTH);
input clk;
logic [FIFO_WIDTH-1:0] data_in;
logic rst_n, wr_en, rd_en;
logic [FIFO_WIDTH-1:0] data_out;
logic wr_ack, overflow,underflow;
logic full, empty, almostfull, almostempty ;



modport DUT (
input data_in,
input clk, rst_n, wr_en, rd_en,

output data_out,
output wr_ack, overflow,underflow,
output full, empty, almostfull, almostempty
);

endinterface
```

6- FIFO_test

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

`uvm_component_utils (FIFO_test_class)

FIFO_env_class env;
FIFO_config_class FIFO_cnfg;

FIFO_sequence_reset reset_seq;
write_only_sequence write_only_seq;
read_only_sequence read_only_seq;
write_read_sequence write_read_seq;

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

endfunction

function void build_phase(uvm_phase phase);
  super.build_phase(phase);
    env = FIFO_env_class :: type_id :: create ("env", this);

    FIFO_cnfg = FIFO_config_class :: type_id :: create ("FIFO_cnfg");

    reset_seq = FIFO_sequence_reset :: type_id :: create ("reset_seq");

    write_only_seq = write_only_sequence :: type_id :: create ("write_only_seq");
    read_only_seq  = read_only_sequence  :: type_id :: create ("read_only_seq");
    write_read_seq = write_read_sequence :: type_id :: create ("write_read_seq");

    if(!uvm_config_db#(virtual FIFO_Interface) :: get(this,"","FIFO_if",FIFO_cnfg.FIFO_vif))
    `uvm_fatal ("build_phas","test - unable to get the virtual interface of the FIFO from uvm_config_db ");

    uvm_config_db #(FIFO_config_class) :: set (this,"*","CFG",FIFO_cnfg);

endfunction

task run_phase(uvm_phase phase);
    super.run_phase(phase);
    phase.raise_objection(this);
    `uvm_info("run_phase","RST is asserted",UVM_LOW)
    reset_seq.start(env.agt.sqr);
    `uvm_info("run_phase","RST is deasserted",UVM_LOW)

    `uvm_info("run_phase","stimulus generation write_only_seq started",UVM_LOW)
    write_only_seq.start(env.agt.sqr);
    `uvm_info("run_phase","stimulus generation write_only_seq ended",UVM_LOW)


    `uvm_info("run_phase","stimulus generation read_only_seq started",UVM_LOW)
    read_only_seq.start(env.agt.sqr);
    `uvm_info("run_phase","stimulus generation read_only_seq ended",UVM_LOW)

    `uvm_info("run_phase","stimulus generation write_read_seq started",UVM_LOW)
    write_read_seq.start(env.agt.sqr);
    `uvm_info("run_phase","stimulus generation write_read_seq ended",UVM_LOW)


    phase.drop_objection(this);
endtask : run_phase

endclass : FIFO_test_class

endpackage : FIFO_test_pkg
```

7- FIFO_env

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

8- FIFO_Driver

```
1  package FIFO_driver_pkg;
2  import uvm_pkg::*;
3  `include "uvm_macros.svh"
4  import FIFO_config_pkg::*;
5  import FIFO_seq_item_pkg::*;
6
7  class FIFO_driver_class extends uvm_driver #(FIFO_seq_item);
8
9  `uvm_component_utils (FIFO_driver_class)
10 virtual FIFO_Interface FIFO_vif;
11 FIFO_config_class FIFO_cnfg;
12
13 FIFO_seq_item stim_seq_item;
14
15 function new (string name ="FIFO_driver_class", uvm_component parent = null);
16   super.new (name,parent);
17 endfunction
18
19 task run_phase(uvm_phase phase);
20   super.run_phase(phase);
21   forever begin
22     stim_seq_item= FIFO_seq_item :: type_id :: create("stim_seq_item");
23     seq_item_port.get_next_item(stim_seq_item);
24
25     FIFO_vif.data_in      = stim_seq_item.data_in;
26     FIFO_vif.rst_n        = stim_seq_item.rst_n;
27     FIFO_vif.wr_en        = stim_seq_item.wr_en;
28     FIFO_vif.rd_en        = stim_seq_item.rd_en;
29
30     @(negedge FIFO_vif.clk );
31     seq_item_port.item_done();
32     `uvm_info ("run_phas",stim_seq_item.convert2string_stimulus(),UVM_HIGH);
33   end
34
35 endtask : run_phase
36
37
38 endclass
39
40
41 endpackage
```

9- FIFO_Config_obj

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

10- FIFO_Sequencer

```
1  package FIFO_sequencer_pkg;
2  import uvm_pkg::*;
3  `include "uvm_macros.svh"
4  import FIFO_seq_item_pkg::*;
5  class FIFO_sequencer extends uvm_sequencer#(FIFO_seq_item);
6  `uvm_component_utils (FIFO_sequencer)
7      function new (string name ="FIFO_sequencer", uvm_component parent = null);
8    super.new (name,parent);
9
10 endfunction
11 endclass
12
13 endpackage
```

11- FIFO_Sequence

I Splitted the FIFO main sequence into multiple sequences as required.

```
package FIFO_sequence_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
import FIFO_seq_item_pkg::*;
class FIFO_sequence_reset extends uvm_sequence #(FIFO_seq_item);

`uvm_object_utils (FIFO_sequence_reset)
FIFO_seq_item seq_item;

function new (string name ="FIFO_sequence_reset");
  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.data_in=0;
  seq_item.wr_en=0;
  seq_item.rd_en=0;
  finish_item(seq_item);
endtask

endclass


class write_only_sequence extends uvm_sequence #(FIFO_seq_item);

`uvm_object_utils (write_only_sequence)
FIFO_seq_item seq_item;

function new (string name ="write_only_sequence");
  super.new (name);

endfunction

task body;
// write only sequence
repeat (10) begin
  seq_item =   FIFO_seq_item :: type_id :: create ("seq_item");
  start_item (seq_item);
  assert (seq_item.randomize() with {seq_item.wr_en==1; seq_item.rd_en==0;});
  finish_item(seq_item);
end

endtask
endclass

class read_only_sequence extends uvm_sequence #(FIFO_seq_item);

`uvm_object_utils (read_only_sequence)
FIFO_seq_item seq_item;

function new (string name ="read_only_sequence");
  super.new (name);

endfunction

task body;
// read only sequence
repeat (10) begin
  seq_item =   FIFO_seq_item :: type_id :: create ("seq_item");
  start_item (seq_item);
  assert (seq_item.randomize() with {seq_item.wr_en==0; seq_item.rd_en==1;});
  finish_item(seq_item);
end

endtask
endclass

class write_read_sequence extends uvm_sequence #(FIFO_seq_item);

`uvm_object_utils (write_read_sequence)
FIFO_seq_item seq_item;

function new (string name ="write_read_sequence");
  super.new (name);

endfunction

task body;

// write read sequence
repeat (10_000) begin
  seq_item =   FIFO_seq_item :: type_id :: create ("seq_item");
  start_item (seq_item);
  assert (seq_item.randomize());
  finish_item(seq_item);
end

endtask
endclass
endpackage
```

12- FIFO_Seq_item

```
1  package FIFO_seq_item_pkg;
2  import uvm_pkg::*;
3  `include "uvm_macros.svh"
4  class FIFO_seq_item extends uvm_sequence_item;
5
6  `uvm_object_utils (FIFO_seq_item)
7  parameter FIFO_WIDTH = 16;
8  parameter FIFO_DEPTH = 8;
9
10 rand bit [FIFO_WIDTH-1:0] data_in;
11 rand bit rst_n, wr_en, rd_en;
12
13 bit [FIFO_WIDTH-1:0] data_out;
14 bit wr_ack, overflow,underflow;
15 bit full, empty, almostfull, almostempty ;
16
17 int RD_EN_ON_DIST=30,WR_EN_ON_DIST=70;
18 function new (string name ="FIFO_seq_item");
19   super.new (name);
20
21 endfunction
22
23 constraint RST {
24     rst_n dist {0:=2, 1:=98};
25 }
26
27 constraint write_enable_dist {
28     wr_en dist {1:=WR_EN_ON_DIST, 0:=(100-WR_EN_ON_DIST)};
29 }
30
31 constraint read_enable_dist {
32     rd_en dist {1:=RD_EN_ON_DIST, 0:=(100-RD_EN_ON_DIST)};
33 }
34
```

```
function string convert2string ();
    return $sformatf("%s rst_n=0b%0b ,data_in=0h%0h ,wr_en=0b%0b, rd_en=0b%0b ,data_out=0h%0h, wr_ack=0b%0b, overflow=0b%0b, underflow=0b%0b , fu
endfunction


function string convert2string_stimulus ();
    return $sformatf("rst_n=0b%0b ,data_in=0h%0h ,wr_en=0b%0b, rd_en=0b%0b ,data_out=0h%0h, wr_ack=0b%0b, overflow=0b%0b, underflow=0b%0b , full=
endfunction

endclass

endpackage
```

13- FIFO_Scoreboard

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

class FIFO_score extends uvm_scoreboard;
    `uvm_component_utils (FIFO_score)

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

parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;

bit [FIFO_WIDTH-1:0] data_out_ref;
bit wr_ack_ref, overflow_ref,underflow_ref;
bit full_ref, empty_ref, almostfull_ref, almostempty_ref ;
int queue_check[$];

int error_count =0;
int correct_count =0;

function new (string name ="FIFO_score", 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(seq_item);
        ref_model(seq_item);
            if(seq_item.data_out !== data_out_ref || seq_item.wr_ack !== wr_ack_ref|| seq_item.overflow !== overflow_ref|| seq_item.underflow !==
                `uvm_error("run_phase",$sformatf("Comparsion failed, Transaction received by the dut: %s while the data_out_ref: 0b%0h  while the
                error_count++;
            end
            else begin
                `uvm_info("run_phase",$sformatf("Correct Shift reg out: %s",seq_item.convert2string()),UVM_HIGH);
                correct_count++;
            end
                end
endtask
```

```
task ref_model(FIFO_seq_item seq_item_chk);

if (!seq_item_chk.rst_n) begin
        wr_ack_ref = 0;
        overflow_ref = 0;
    end
    else if (seq_item_chk.wr_en && !full_ref) begin
        queue_check.push_front(seq_item_chk.data_in);
        wr_ack_ref = 1;
    overflow_ref = 0;
    end
    else begin
        wr_ack_ref = 0;
        if (full_ref & seq_item_chk.wr_en)
            overflow_ref = 1;
        else
            overflow_ref = 0;
    end

    if (!seq_item_chk.rst_n) begin
        underflow_ref=0;
    queue_check.delete();
    end
    else if (seq_item_chk.rd_en && !empty_ref) begin
        data_out_ref = queue_check.pop_back();
    underflow_ref=0;
    end
    else begin
        if (empty_ref && seq_item_chk.rd_en) begin
            underflow_ref =1;
        end
        else begin
            underflow_ref =0;
        end
    end


    full_ref        = (queue_check.size() == seq_item_chk.FIFO_DEPTH)? 1 : 0;
    empty_ref       = (queue_check.size() == 0)? 1 : 0;
    almostfull_ref  = (queue_check.size() == seq_item_chk.FIFO_DEPTH-1)? 1 : 0;
    almostempty_ref = (queue_check.size() == 1)? 1 : 0;


endtask

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

endclass

endpackage
```

14- FIFO_Monitor

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

class FIFO_mon extends uvm_monitor;

`uvm_component_utils (FIFO_mon)
virtual FIFO_Interface FIFO_vif;
FIFO_seq_item rsp_seq_item;

uvm_analysis_port #(FIFO_seq_item) mon_ap;

function new (string name ="FIFO_mon", 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_seq_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.overflow    = FIFO_vif.overflow;
      rsp_seq_item.underflow   = FIFO_vif.underflow;
      rsp_seq_item.full        = FIFO_vif.full;
      rsp_seq_item.empty       = FIFO_vif.empty;
      rsp_seq_item.almostfull  = FIFO_vif.almostfull;
      rsp_seq_item.almostempty = FIFO_vif.almostempty;

    mon_ap.write(rsp_seq_item);
    `uvm_info ("run_phas",rsp_seq_item.convert2string_stimulus(),UVM_HIGH);
    end
endtask : run_phase

endclass
endpackage
```

15- FIFO_Coverage

Only I ignore bins which impossible to happen

Like: -

when wr_en=0 it’s impossible to make wr_ak=1

when wr_en=0 it’s impossible to make overflow=1

when rd_en =0 it’s impossible to make underflow=1

when rd_en =1 it’s impossible to make full=1

when wr_en=1 it’s impossible to make empty=1

```
package FIFO_coverage_pkg;
import FIFO_seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class FIFO_coverage extends uvm_component;

  `uvm_component_utils(FIFO_coverage);
    FIFO_seq_item seq_item;
    uvm_analysis_export #(FIFO_seq_item) cov_export;
    uvm_tlm_analysis_fifo #(FIFO_seq_item) cov_fifo;


covergroup g1;

// EXTRA will cover rst also
      RST           : coverpoint  seq_item.rst_n;
      wr_en         : coverpoint  seq_item.wr_en;
      rd_en         : coverpoint  seq_item.rd_en;
      wr_ack        : coverpoint  seq_item.wr_ack;
      overflow      : coverpoint  seq_item.overflow;
      underflow     : coverpoint  seq_item.underflow;
      full          : coverpoint  seq_item.full;
      empty         : coverpoint  seq_item.empty;
      almostfull    : coverpoint  seq_item.almostfull;
      almostempty   : coverpoint  seq_item.almostempty;

        wr_en_rd_en_wr_ack      : cross wr_en , rd_en , wr_ack  iff (seq_item.rst_n)
        {
          ignore_bins ignore_bin1  =  binsof(wr_en)  intersect {0} && binsof(wr_ack)    intersect {1};

        }  // only if rst deasseted
        wr_en_rd_en_overflow    : cross wr_en , rd_en , overflow iff (seq_item.rst_n)
        {
          ignore_bins ignore_bin2  =  binsof(wr_en)  intersect {0} &&  binsof(overflow) intersect {1};
        }// only if rst deasseted
        wr_en_rd_en_underflow   : cross wr_en , rd_en , underflow iff (seq_item.rst_n)
        {
          ignore_bins ignore_bin3  =  binsof(rd_en)  intersect {0} && binsof(underflow) intersect {1};
        } // only if rst deasseted
        wr_en_rd_en_full        : cross wr_en , rd_en , full     iff (seq_item.rst_n)
        {
          ignore_bins ignore_bin4 =   binsof(rd_en)  intersect {1} && binsof(full)      intersect {1};
        } // only if rst deasseted
        wr_en_rd_en_empty       : cross wr_en , rd_en , empty    iff (seq_item.rst_n)
        {
          ignore_bins ignore_bin5 =  binsof(wr_en)  intersect {1} && binsof(empty)     intersect {1};
        } // only if rst deasseted
        wr_en_rd_en_almostfull  : cross wr_en , rd_en , almostfull   iff (seq_item.rst_n) ; // only if rst deasseted
        wr_en_rd_en_almostempty : cross wr_en , rd_en , almostempty  iff (seq_item.rst_n) ; // only if rst deasseted
        endgroup

function new (string name ="FIFO_coverage", uvm_component parent = null);
    super.new (name,parent);
    g1=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);
      g1.sample();
    end
endtask
endclass

endpackage
```

16- FIFO_Agent

```
package FIFO_agent_pkg;

import uvm_pkg::*;
`include "uvm_macros.svh" ;
import FIFO_sequencer_pkg::*;
import FIFO_driver_pkg::*;
import FIFO_config_pkg::*;
import FIFO_mon_pkg::*;
import FIFO_seq_item_pkg::*;

class FIFO_agent extends uvm_agent;

`uvm_component_utils (FIFO_agent)

FIFO_sequencer sqr;
FIFO_driver_class driver;
FIFO_config_class FIFO_cnfg;
FIFO_mon mon;

uvm_analysis_port #(FIFO_seq_item) agt_ap;

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

endfunction

function void build_phase(uvm_phase phase);
    super.build_phase(phase);
if(!uvm_config_db#(FIFO_config_class) :: get(this,"","CFG",FIFO_cnfg))
    `uvm_fatal ("build_phas","unable to get configuration objecet ");
driver = FIFO_driver_class :: type_id :: create ("driver",this);
sqr    = FIFO_sequencer :: type_id :: create ("sqr",this);
mon    = FIFO_mon :: type_id :: create ("mon",this);
agt_ap = new ("agt_ap",this);

endfunction

function void connect_phase (uvm_phase phase);
  super.connect_phase(phase)  ;
  driver.FIFO_vif=FIFO_cnfg.FIFO_vif;
  mon.FIFO_vif=FIFO_cnfg.FIFO_vif;
  driver.seq_item_port.connect(sqr.seq_item_export);
  mon.mon_ap.connect(agt_ap);

endfunction
endclass



endpackage
```

17- Assertions

The required Assertions are outputs flags and the counter only

I did extra Assertions

Assertions: -

1- 1 Assertion for max_fifo_addr (EXTRA)
2- 2 Assertions for wr_ptr (EXTRA)
3- 2 Assertions for rd_ptr (EXTRA)
4- 5 Assertions for Count
5- 3 Assertions for wr_ack
6- 3 Assertions for overflow
7- 3 Assertions for underflow
8- 2 Assertions for full
9- 2 Assertions for almostfull
10- 2 Assertions for Relation between full and almostfull (EXTRA)
11- 2 Assertions for empty
12- 2 Assertions for almostempty
13- 2 Assertions for Relation between empty and almostempty (EXTRA)

And I bind it in top module as required

Feature	**Assertion**
Check max_fifo_addr	assert final (max_fifo_addr == $clog2(FIFO_IF.FIFO_DEPTH))
If rst asserted wr_ptr=0	if(!FIFO_IF.rst_n) wr_ptr1_p: assert final (wr_ptr == 0);
If rst deassetred and wr_en and not full wr_ptr will increment	@(posedge FIFO_IF.clk) disable iff (!FIFO_IF.rst_n) (FIFO_IF.wr_en && count < FIFO_IF.FIFO_DEPTH) |-> ##1 (wr_ptr == ($past(wr_ptr)+1'b1));
If rst asserted rd_ptr=0	if(!FIFO_IF.rst_n) rd_ptr1_p: assert final (rd_ptr == 0);
If rst deassetred and rd_en and not empty rd_ptr will increment	@(posedge FIFO_IF.clk) disable iff (!FIFO_IF.rst_n) (FIFO_IF.rd_en && count != 0) |-> ##1 (rd_ptr == ($past(rd_ptr)+1'b1));
If rst asserted count=0	if(!FIFO_IF.rst_n) counter1_p: assert final (count == 0);
If rst deassetred and wr_en and not rd_en and not full count will increment	@(posedge FIFO_IF.clk) disable iff (!FIFO_IF.rst_n) (({FIFO_IF.wr_en, FIFO_IF.rd_en} == 2'b10) && !FIFO_IF.full) |-> ##1 (count == ($past(count)+1'b1))
If rst deassetred and not wr_en and rd_en and not empty count will deccrement	@(posedge FIFO_IF.clk) disable iff (!FIFO_IF.rst_n) (({FIFO_IF.wr_en, FIFO_IF.rd_en} == 2'b01) && !FIFO_IF.empty) |-> ##1 (count == ($past(count)-1'b1))
If rst deassetred and wr_en and rd_en and full count will deccrement	@(posedge FIFO_IF.clk) disable iff (!FIFO_IF.rst_n) (({FIFO_IF.wr_en, FIFO_IF.rd_en} == 2'b11) && FIFO_IF.full) |-> ##1 (count == ($past(count)-1'b1))
If rst deassetred and wr_en and rd_en and empty count will increment	@(posedge FIFO_IF.clk) disable iff (!FIFO_IF.rst_n) (({FIFO_IF.wr_en, FIFO_IF.rd_en} == 2'b11) && FIFO_IF.empty) |-> ##1 (count == ($past(count)+1'b1))

If rst asserted wr_ack=0	if(!FIFO_IF.rst_n) wr_ack1_p: assert final (FIFO_IF.wr_ack == 0);
If rst deassetred and wr_en and not full wr_ack=1	@(posedge FIFO_IF.clk) disable iff (!FIFO_IF.rst_n) (FIFO_IF.wr_en && count < FIFO_IF.FIFO_DEPTH) |-> ##1 (FIFO_IF.wr_ack == 1);
If rst deassetred and wr_en and full wr_ack=0	@(posedge FIFO_IF.clk) disable iff (!FIFO_IF.rst_n) (!FIFO_IF.wr_en || !(count < FIFO_IF.FIFO_DEPTH)) |-> ##1 (FIFO_IF.wr_ack == 0);
If rst asserted overflow=0	if(!FIFO_IF.rst_n) overflow1_p: assert final (FIFO_IF.overflow == 0);
If rst deassetred and wr_en and full overflow =1	@(posedge FIFO_IF.clk) disable iff (!FIFO_IF.rst_n) (FIFO_IF.full && FIFO_IF.wr_en) |-> ##1 (FIFO_IF.overflow == 1);
If rst deassetred and wr_en and not full overflow =0	@(posedge FIFO_IF.clk) disable iff (!FIFO_IF.rst_n) ((!FIFO_IF.full) && FIFO_IF.wr_en) |-> ##1 (FIFO_IF.overflow == 0);
If rst asserted underflow=0	if(!FIFO_IF.rst_n) underflow1_p: assert final (FIFO_IF.underflow == 0);
If rst deassetred and rd_en and empty underflow =1	@(posedge FIFO_IF.clk) disable iff (!FIFO_IF.rst_n) (FIFO_IF.empty && FIFO_IF.rd_en) |-> ##1 (FIFO_IF.underflow == 1);
If rst deassetred and rd_en and not empty underflow =0	@(posedge FIFO_IF.clk) disable iff (!FIFO_IF.rst_n) ((!FIFO_IF.empty) && FIFO_IF.rd_en) |-> ##1 (FIFO_IF.underflow == 0);
If count =FIFO_DEPTH full=1	@(posedge FIFO_IF.clk) (count == FIFO_IF.FIFO_DEPTH) |-> (FIFO_IF.full == 1);
If count! =FIFO_DEPTH full=0	@(posedge FIFO_IF.clk) (count != FIFO_IF.FIFO_DEPTH) |-> (FIFO_IF.full == 0);
If count =FIFO_DEPTH-1 almostfull=1	@(posedge FIFO_IF.clk) (count == (FIFO_IF.FIFO_DEPTH-1'b1)) |-> (FIFO_IF.almostfull == 1);
If count! =FIFO_DEPTH-1 almostfull=0	@(posedge FIFO_IF.clk) (count != (FIFO_IF.FIFO_DEPTH-1'b1)) |-> (FIFO_IF.almostfull == 0);
If rst deasserted if almostfull and not rd_en and wr_en next cycle will be full	@(posedge FIFO_IF.clk) disable iff (!FIFO_IF.rst_n) (FIFO_IF.almostfull && (!(FIFO_IF.rd_en)&&(FIFO_IF.wr_en))) |-> ##1 (FIFO_IF.full);
If rst deasserted if full and rd_en and not wr_en next cycle will be almostfull	@(posedge FIFO_IF.clk) disable iff (!FIFO_IF.rst_n) (FIFO_IF.full && ((FIFO_IF.rd_en)&&(!FIFO_IF.wr_en))) |-> ##1 (FIFO_IF.almostfull);
If count =0 empty=1	@(posedge FIFO_IF.clk) (count == 0) |-> (FIFO_IF.empty == 1);
If count! =0 empty =0	@(posedge FIFO_IF.clk) (count != 0) |-> (FIFO_IF.empty == 0);
If count =1 almostempty=1	@(posedge FIFO_IF.clk) (count == 1) |-> (FIFO_IF.almostempty == 1);
If count! =1 almostempty=0	@(posedge FIFO_IF.clk) (count != 1) |-> (FIFO_IF.almostempty == 0);
If rst deasserted if almostempty and rd_en and not wr_en next cycle will be empty	@(posedge FIFO_IF.clk) disable iff (!FIFO_IF.rst_n) (FIFO_IF.almostempty && (!(FIFO_IF.wr_en)&& (FIFO_IF.rd_en))) |-> ##1 (FIFO_IF.empty);
If rst deasserted if empty and not rd_en and wr_en next cycle will be almostempty	@(posedge FIFO_IF.clk) disable iff (!FIFO_IF.rst_n) (FIFO_IF.empty && ((FIFO_IF.wr_en)&& (!FIFO_IF.rd_en))) |-> ##1 (FIFO_IF.almostempty);

```
module Assertions(FIFO_Interface.DUT FIFO_IF);

localparam max_fifo_addr = DUT.max_fifo_addr;
logic [max_fifo_addr-1:0] wr_ptr, rd_ptr;
logic [max_fifo_addr:0] count;

assign wr_ptr = DUT.wr_ptr;
assign rd_ptr = DUT.rd_ptr;
assign count  = DUT.count;
//EXTA
// max_fifo_addr Assertions
always_comb begin
    max_fifo_addr_p: assert final (max_fifo_addr == $clog2(FIFO_IF.FIFO_DEPTH));
end
//EXTA
// wr_ptr Assertions
//immediate because the rst is async
always_comb begin
    if(!FIFO_IF.rst_n)
    wr_ptr1_p: assert final (wr_ptr == 0);
end

property wr_ptr2;
 @(posedge FIFO_IF.clk) disable iff (!FIFO_IF.rst_n)  (FIFO_IF.wr_en && count < FIFO_IF.FIFO_DEPTH) |-> ##1 (wr_ptr == ($past(wr_ptr)+1'b1));
endproperty
wr_ptr2_p: assert property (wr_ptr2);
wr_ptr2_c: cover  property (wr_ptr2);
//EXTA
// rd_ptr Assertions
//immediate because the rst is async
always_comb begin
    if(!FIFO_IF.rst_n)
    rd_ptr1_p: assert final (rd_ptr == 0);
end

property rd_ptr2;
 @(posedge FIFO_IF.clk) disable iff (!FIFO_IF.rst_n)  (FIFO_IF.rd_en && count != 0) |-> ##1 (rd_ptr == ($past(rd_ptr)+1'b1));
endproperty
rd_ptr2_p: assert property (rd_ptr2);
rd_ptr2_c: cover  property (rd_ptr2);

```

```
// Counter Assertions
//immediate because the rst is async
always_comb begin
    if(!FIFO_IF.rst_n)
    counter1_p: assert final (count == 0);
end

property counter2;
 @(posedge FIFO_IF.clk) disable iff (!FIFO_IF.rst_n) (({FIFO_IF.wr_en, FIFO_IF.rd_en} == 2'b10) && !FIFO_IF.full) |-> ##1 (count == ($past(count)+1'b1))
endproperty
counter2_p: assert property (counter2);
counter2_c: cover  property (counter2);


property counter3;
 @(posedge FIFO_IF.clk) disable iff (!FIFO_IF.rst_n) (({FIFO_IF.wr_en, FIFO_IF.rd_en} == 2'b01) && !FIFO_IF.empty) |-> ##1 (count == ($past(count)-1'b1))
endproperty
counter3_p: assert property (counter3);
counter3_c: cover  property (counter3);

property counter4;
 @(posedge FIFO_IF.clk) disable iff (!FIFO_IF.rst_n) (({FIFO_IF.wr_en, FIFO_IF.rd_en} == 2'b11) && FIFO_IF.full) |-> ##1 (count == ($past(count)-1'b1))
endproperty
counter4_p: assert property (counter4);
counter4_c: cover  property (counter4);

property counter5;
 @(posedge FIFO_IF.clk) disable iff (!FIFO_IF.rst_n) (({FIFO_IF.wr_en, FIFO_IF.rd_en} == 2'b11) && FIFO_IF.empty) |-> ##1 (count == ($past(count)+1'b1))
endproperty
counter5_p: assert property (counter5);
counter5_c: cover  property (counter5);
```

```
// wr_ack Assertions
//immediate because the rst is async
always_comb begin
    if(!FIFO_IF.rst_n)
    wr_ack1_p: assert final (FIFO_IF.wr_ack == 0);
end

property wr_ack2;
 @(posedge FIFO_IF.clk) disable iff (!FIFO_IF.rst_n) (FIFO_IF.wr_en && count < FIFO_IF.FIFO_DEPTH)  |-> ##1 (FIFO_IF.wr_ack == 1);
endproperty
wr_ack2_p: assert property (wr_ack2);
wr_ack2_c: cover  property (wr_ack2);

property wr_ack3;
 @(posedge FIFO_IF.clk) disable iff (!FIFO_IF.rst_n) (!FIFO_IF.wr_en || !(count < FIFO_IF.FIFO_DEPTH))  |-> ##1 (FIFO_IF.wr_ack == 0);
endproperty
wr_ack3_p: assert property (wr_ack3);
wr_ack3_c: cover  property (wr_ack3);

//overflow Assertions
//immediate because the rst is async
always_comb begin
    if(!FIFO_IF.rst_n)
    overflow1_p: assert final (FIFO_IF.overflow == 0);
end

property overflow2;
 @(posedge FIFO_IF.clk) disable iff (!FIFO_IF.rst_n) (FIFO_IF.full && FIFO_IF.wr_en) |-> ##1 (FIFO_IF.overflow == 1);
endproperty
overflow2_p: assert property (overflow2);
overflow2_c: cover  property (overflow2);

property overflow3;
 @(posedge FIFO_IF.clk) disable iff (!FIFO_IF.rst_n) ((!FIFO_IF.full) && FIFO_IF.wr_en) |-> ##1 (FIFO_IF.overflow == 0);
endproperty
overflow3_p: assert property (overflow3);
overflow3_c: cover  property (overflow3);
```

```
//underflow Assertions
//immediate because the rst is async
always_comb begin
    if(!FIFO_IF.rst_n)
    underflow1_p: assert final (FIFO_IF.underflow == 0);
end

property underflow2;
 @(posedge FIFO_IF.clk) disable iff (!FIFO_IF.rst_n) (FIFO_IF.empty && FIFO_IF.rd_en) |-> ##1 (FIFO_IF.underflow == 1);
endproperty
underflow2_p: assert property (underflow2);
underflow2_c: cover  property (underflow2);

property underflow3;
 @(posedge FIFO_IF.clk) disable iff (!FIFO_IF.rst_n) ((!FIFO_IF.empty) && FIFO_IF.rd_en) |-> ##1 (FIFO_IF.underflow == 0);
endproperty
underflow3_p: assert property (underflow3);
underflow3_c: cover  property (underflow3);


// full Assertions
property full1;
 @(posedge FIFO_IF.clk)  (count == FIFO_IF.FIFO_DEPTH)  |->  (FIFO_IF.full == 1);
endproperty
full1_p: assert property (full1);
full1_c: cover  property (full1);

property full2;
 @(posedge FIFO_IF.clk)  (count != FIFO_IF.FIFO_DEPTH)  |->  (FIFO_IF.full == 0);
endproperty
full2_p: assert property (full2);
full2_c: cover  property (full2);
```

```
// almostfull Assertions
property almostfull1;
 @(posedge FIFO_IF.clk)  (count == (FIFO_IF.FIFO_DEPTH-1'b1))  |->  (FIFO_IF.almostfull == 1);
endproperty
almostfull1_p: assert property (almostfull1);
almostfull1_c: cover  property (almostfull1);

property almostfull2;
 @(posedge FIFO_IF.clk)  (count != (FIFO_IF.FIFO_DEPTH-1'b1))  |->  (FIFO_IF.almostfull == 0);
endproperty
almostfull2_p: assert property (almostfull2);
almostfull2_c: cover  property (almostfull2);
//EXTRA
// compination assertions between full and almostfull
property full_almostfull1;
 @(posedge FIFO_IF.clk) disable iff (!FIFO_IF.rst_n) (FIFO_IF.almostfull && (!(FIFO_IF.rd_en)&&(FIFO_IF.wr_en)) )  |-> ##1  (FIFO_IF.full);
endproperty
full_almostfull1_p: assert property (full_almostfull1);
full_almostfull1_c: cover  property (full_almostfull1);


property full_almostfull2;
 @(posedge FIFO_IF.clk) disable iff (!FIFO_IF.rst_n) (FIFO_IF.full && ((FIFO_IF.rd_en)&&(!FIFO_IF.wr_en)) )  |-> ##1  (FIFO_IF.almostfull);
endproperty
full_almostfull2_p: assert property (full_almostfull2);
full_almostfull2_c: cover  property (full_almostfull2);

```

```
// empty Assertions
property empty1;
 @(posedge FIFO_IF.clk)  (count == 0)  |->  (FIFO_IF.empty == 1);
endproperty
empty1_p: assert property (empty1);
empty1_c: cover  property (empty1);

property empty2;
 @(posedge FIFO_IF.clk)  (count != 0)  |->  (FIFO_IF.empty == 0);
endproperty
empty2_p: assert property (empty2);
empty2_c: cover  property (empty2);


// almostempty Assertions
property almostempty1;
 @(posedge FIFO_IF.clk)  (count == 1)  |->  (FIFO_IF.almostempty == 1);
endproperty
almostempty1_p: assert property (almostempty1);
almostempty1_c: cover  property (almostempty1);

property almostempty2;
 @(posedge FIFO_IF.clk)  (count != 1)  |->  (FIFO_IF.almostempty == 0);
endproperty
almostempty2_p: assert property (almostempty2);
almostempty2_c: cover  property (almostempty2);

//EXTRA
// compination assertions between empty and almostempty
property empty_almostempty1;
 @(posedge FIFO_IF.clk) disable iff (!FIFO_IF.rst_n) (FIFO_IF.almostempty && (!(FIFO_IF.wr_en)&& (FIFO_IF.rd_en)) )  |-> ##1  (FIFO_IF.empty);
endproperty
empty_almostempty1_p: assert property (empty_almostempty1);
empty_almostempty1_c: cover  property (empty_almostempty1);


property empty_almostempty2;
 @(posedge FIFO_IF.clk) disable iff (!FIFO_IF.rst_n) (FIFO_IF.empty && ((FIFO_IF.wr_en)&& (!FIFO_IF.rd_en)) )  |-> ##1  (FIFO_IF.almostempty);
endproperty
empty_almostempty2_p: assert property (empty_almostempty2);
empty_almostempty2_c: cover  property (empty_almostempty2);

endmodule
```

18- Code Coverage

Branch Coverage

```
================================================================================
=== Instance: /top/DUT
=== Design Unit: work.FIFO
================================================================================
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        25        25         0   100.00%
```

Statement Coverage

```
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                      29        29         0   100.00%
```

Toggle Coverage

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

Also, I will attach the complete report in zip file if you want to see more details about coverage.

19- Functional Coverage

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

Covergroup Coverage:
    Covergroups                 1        na        na   100.00%
        Coverpoints/Crosses    17        na        na        na
            Covergroup Bins    66        66         0   100.00%
```

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

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

Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment
/FIFO_coverage_pkg/FIFO_coverage		100.00%							
TYPE g1		100.00%	100	100.00...			auto(1)		
CVP g1::RST		100.00%	100	100.00...					
CVP g1::wr_en		100.00%	100	100.00...					
CVP g1::rd_en		100.00%	100	100.00...					
CVP g1::wr_ack		100.00%	100	100.00...					
CVP g1::overflow		100.00%	100	100.00...					
CVP g1::underflow		100.00%	100	100.00...					
CVP g1::full		100.00%	100	100.00...					
CVP g1::empty		100.00%	100	100.00...					
CVP g1::almostfull		100.00%	100	100.00...					
CVP g1::almostempty		100.00%	100	100.00...					
CROSS g1::wr_en_rd_en_wr_ack		100.00%	100	100.00...					
CROSS g1::wr_en_rd_en_overflow		100.00%	100	100.00...					
CROSS g1::wr_en_rd_en_underflow		100.00%	100	100.00...					
CROSS g1::wr_en_rd_en_full		100.00%	100	100.00...					
CROSS g1::wr_en_rd_en_empty		100.00%	100	100.00...					
CROSS g1::wr_en_rd_en_almostfull		100.00%	100	100.00...					
CROSS g1::wr_en_rd_en_almostempty		100.00%	100	100.00...					

Also, I will attach the complete report in zip file if you want to see more details about coverage.

20- Assertions Coverage

```
================================================================================
=== Instance: /top/DUT/Assertions_inst
=== Design Unit: work.Assertions
================================================================================

Directive Coverage:
    Directives                  24        24         0   100.00%
```

```
TOTAL DIRECTIVE COVERAGE: 100.00%  COVERS: 24

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

```
================================================================================
=== Instance: /top/DUT/Assertions_inst
=== Design Unit: work.Assertions
================================================================================

Assertion Coverage:
    Assertions                      31         31          0   100.00%
--------------------------------------------------------------------
```

Name	Language	Enabled	Log	Count	AtLe	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
/top/DUT/Assertions_inst/wr_ptr2_c	SVA	✓	Off	415	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/Assertions_inst/rd_ptr2_c	SVA	✓	Off	295	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/Assertions_inst/counter2_c	SVA	✓	Off	291	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/Assertions_inst/counter3_c	SVA	✓	Off	92	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/Assertions_inst/counter4_c	SVA	✓	Off	86	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/Assertions_inst/counter5_c	SVA	✓	Off	7	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/Assertions_inst/wr_ack2_c	SVA	✓	Off	415	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/Assertions_inst/wr_ack3_c	SVA	✓	Off	570	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/Assertions_inst/overflow2_c	SVA	✓	Off	282	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/Assertions_inst/overflow3_c	SVA	✓	Off	415	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/Assertions_inst/underflow2_c	SVA	✓	Off	12	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/Assertions_inst/underflow3_c	SVA	✓	Off	295	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/Assertions_inst/full1_c	SVA	✓	Off	393	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/Assertions_inst/full2_c	SVA	✓	Off	628	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/Assertions_inst/almostfull1_c	SVA	✓	Off	272	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/Assertions_inst/almostfull2_c	SVA	✓	Off	749	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/Assertions_inst/full_almostfull1_c	SVA	✓	Off	123	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/Assertions_inst/full_almostfull2_c	SVA	✓	Off	28	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/Assertions_inst/empty1_c	SVA	✓	Off	48	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/Assertions_inst/empty2_c	SVA	✓	Off	973	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/Assertions_inst/almostempty1_c	SVA	✓	Off	46	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/Assertions_inst/almostempty2_c	SVA	✓	Off	975	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/Assertions_inst/empty_almostempty1_c	SVA	✓	Off	4	1	Unlimited	1	100%		✓	0	0	0 ns	0
/top/DUT/Assertions_inst/empty_almostempty2_c	SVA	✓	Off	15	1	Unlimited	1	100%		✓	0	0	0 ns	0

/top/DUT/Assertions_inst/max_fifo_addr_p	Immediate	SVA	or	0	1	-	-	-	-	off	assert (3==$clog2(FIFO_IF.FIFO_DEPTH))	✓
/top/DUT/Assertions_inst/wr_ptr1_p	Immediate	SVA	or	0	1	-	-	-	-	off	assert (wr_ptr==0)	✓
/top/DUT/Assertions_inst/wr_ptr2_p	Concurrent	SVA	or	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge FIFO_IF.clk) disable iff (~FIFO_IF.rst_n) (((FIFO_IF.wr_en…	✓
/top/DUT/Assertions_inst/rd_ptr1_p	Immediate	SVA	or	0	1	-	-	-	-	off	assert (rd_ptr==0)	✓
/top/DUT/Assertions_inst/rd_ptr2_p	Concurrent	SVA	or	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge FIFO_IF.clk) disable iff (~FIFO_IF.rst_n) (((FIFO_IF.rd_en…	✓
/top/DUT/Assertions_inst/counter1_p	Immediate	SVA	or	0	1	-	-	-	-	off	assert (count==0)	✓
/top/DUT/Assertions_inst/counter2_p	Concurrent	SVA	or	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge FIFO_IF.clk) disable iff (~FIFO_IF.rst_n) (((~FIFO_IF.rd_e…	✓
/top/DUT/Assertions_inst/counter3_p	Concurrent	SVA	or	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge FIFO_IF.clk) disable iff (~FIFO_IF.rst_n) (((FIFO_IF.rd_en…	✓
/top/DUT/Assertions_inst/counter4_p	Concurrent	SVA	or	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge FIFO_IF.clk) disable iff (~FIFO_IF.rst_n) (((FIFO_IF.rd_en…	✓
/top/DUT/Assertions_inst/counter5_p	Concurrent	SVA	or	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge FIFO_IF.clk) disable iff (~FIFO_IF.rst_n) (((FIFO_IF.rd_en…	✓
/top/DUT/Assertions_inst/wr_ack1_p	Immediate	SVA	or	0	1	-	-	-	-	off	assert (~FIFO_IF.wr_ack)	✓
/top/DUT/Assertions_inst/wr_ack2_p	Concurrent	SVA	or	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge FIFO_IF.clk) disable iff (~FIFO_IF.rst_n) (((FIFO_IF.wr_en…	✓
/top/DUT/Assertions_inst/wr_ack3_p	Concurrent	SVA	or	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge FIFO_IF.clk) disable iff (~FIFO_IF.rst_n) (((~FIFO_IF.wr_…	✓
/top/DUT/Assertions_inst/overflow1_p	Immediate	SVA	or	0	1	-	-	-	-	off	assert (~FIFO_IF.overflow)	✓
/top/DUT/Assertions_inst/overflow2_p	Concurrent	SVA	or	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge FIFO_IF.clk) disable iff (~FIFO_IF.rst_n) (((FIFO_IF.full&&…	✓
/top/DUT/Assertions_inst/overflow3_p	Concurrent	SVA	or	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge FIFO_IF.clk) disable iff (~FIFO_IF.rst_n) (((~FIFO_IF.full&…	✓
/top/DUT/Assertions_inst/underflow1_p	Immediate	SVA	or	0	1	-	-	-	-	off	assert (~FIFO_IF.underflow)	✓
/top/DUT/Assertions_inst/underflow2_p	Concurrent	SVA	or	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge FIFO_IF.clk) disable iff (~FIFO_IF.rst_n) (((FIFO_IF.empty…	✓
/top/DUT/Assertions_inst/underflow3_p	Concurrent	SVA	or	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge FIFO_IF.clk) disable iff (~FIFO_IF.rst_n) (((~FIFO_IF.emp…	✓
/top/DUT/Assertions_inst/full1_p	Concurrent	SVA	or	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge FIFO_IF.clk) (count==FIFO_IF.FIFO_DEPTH)|->FIFO_IF.full)	✓
/top/DUT/Assertions_inst/full2_p	Concurrent	SVA	or	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge FIFO_IF.clk) (count!=FIFO_IF.FIFO_DEPTH)|->~FIFO_IF.f…	✓
/top/DUT/Assertions_inst/almostfull1_p	Concurrent	SVA	or	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge FIFO_IF.clk) (count==FIFO_IF.FIFO_DEPTH-1)|->FIFO_IF…	✓
/top/DUT/Assertions_inst/almostfull2_p	Concurrent	SVA	or	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge FIFO_IF.clk) (count!=FIFO_IF.FIFO_DEPTH-1)|->~FIFO_I…	✓
/top/DUT/Assertions_inst/full_almostfull1_p	Concurrent	SVA	or	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge FIFO_IF.clk) disable iff (~FIFO_IF.rst_n) (((FIFO_IF.almos…	✓
/top/DUT/Assertions_inst/full_almostfull2_p	Concurrent	SVA	or	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge FIFO_IF.clk) disable iff (~FIFO_IF.rst_n) (((FIFO_IF.full&&…	✓
/top/DUT/Assertions_inst/empty1_p	Concurrent	SVA	or	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge FIFO_IF.clk) (count==0)|->FIFO_IF.empty)	✓
/top/DUT/Assertions_inst/empty2_p	Concurrent	SVA	or	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge FIFO_IF.clk) (count!=0)|->~FIFO_IF.empty)	✓
/top/DUT/Assertions_inst/almostempty1_p	Concurrent	SVA	or	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge FIFO_IF.clk) (count==1)|->FIFO_IF.almostempty)	✓
/top/DUT/Assertions_inst/almostempty2_p	Concurrent	SVA	or	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge FIFO_IF.clk) (count!=1)|->~FIFO_IF.almostempty)	✓
/top/DUT/Assertions_inst/empty_almostempty1_p	Concurrent	SVA	or	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge FIFO_IF.clk) disable iff (~FIFO_IF.rst_n) (((FIFO_IF.almos…	✓
/top/DUT/Assertions_inst/empty_almostempty2_p	Concurrent	SVA	or	0	1	-	0B	0B	0 ns	0 off	assert(@(posedge FIFO_IF.clk) disable iff (~FIFO_IF.rst_n) (((FIFO_IF.empty…	✓

Also, I will attach the complete report in zip file if you want to see more details about coverage.

21- Do File

```
vlib work
vlog *v +cover -covercells
vsim -voptargs=+acc work.top -cover +UVM_VERBOSITY=UVM_MEDIUM -classdebug -uvmcontrol=all

add wave -position insertpoint sim:/top/FIFO_IF/*

add wave /top/DUT/Assertions_inst/max_fifo_addr_p /top/DUT/Assertions_inst/wr_ptr1_p
add wave /top/DUT/Assertions_inst/wr_ptr2_p /top/DUT/Assertions_inst/rd_ptr1_p
add wave /top/DUT/Assertions_inst/rd_ptr2_p /top/DUT/Assertions_inst/counter1_p
add wave /top/DUT/Assertions_inst/counter2_p /top/DUT/Assertions_inst/counter3_p
add wave /top/DUT/Assertions_inst/counter4_p /top/DUT/Assertions_inst/counter5_p
add wave /top/DUT/Assertions_inst/wr_ack1_p /top/DUT/Assertions_inst/wr_ack2_p
add wave /top/DUT/Assertions_inst/wr_ack3_p /top/DUT/Assertions_inst/overflow1_p
add wave /top/DUT/Assertions_inst/overflow2_p /top/DUT/Assertions_inst/overflow3_p
add wave /top/DUT/Assertions_inst/underflow1_p /top/DUT/Assertions_inst/underflow2_p
add wave /top/DUT/Assertions_inst/underflow3_p /top/DUT/Assertions_inst/full1_p /top/DUT/Assertions_inst/full2_p
add wave /top/DUT/Assertions_inst/almostfull1_p /top/DUT/Assertions_inst/almostfull2_p
add wave /top/DUT/Assertions_inst/full_almostfull1_p /top/DUT/Assertions_inst/full_almostfull2_p
add wave /top/DUT/Assertions_inst/empty1_p /top/DUT/Assertions_inst/empty2_p /top/DUT/Assertions_inst/almostempty1_p
add wave /top/DUT/Assertions_inst/almostempty2_p /top/DUT/Assertions_inst/empty_almostempty1_p
add wave /top/DUT/Assertions_inst/empty_almostempty2_p


coverage save top.ucdb -onexit
run -all
#vcover report top.ucdb -details -annotate -all -output code_coverage_rpt.txt
```

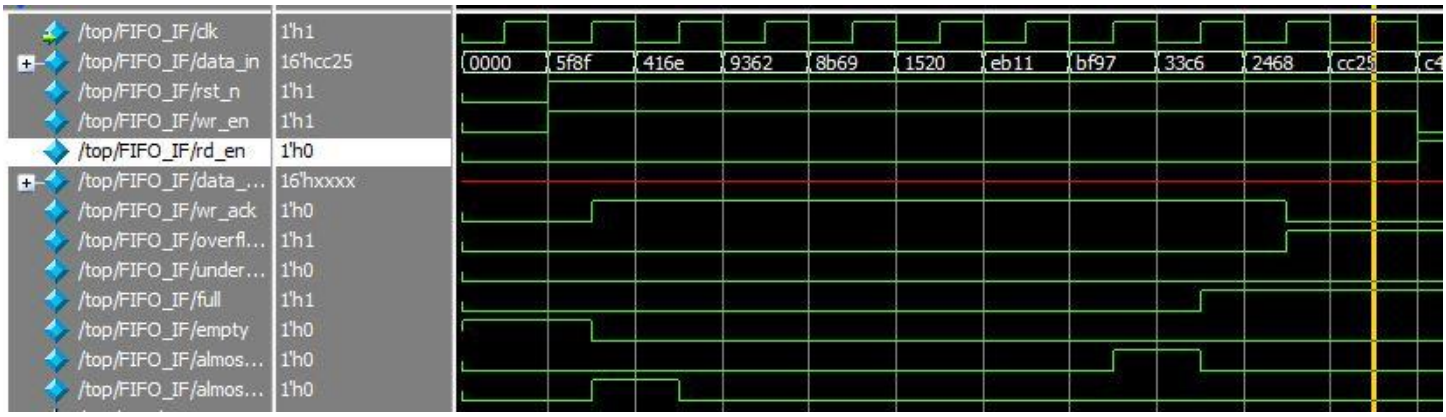
22- Transcript

```
# UVM_INFO FIFO_test.sv(46) @ 0: uvm_test_top [run_phase] RST is asserted
# ********************************************************************************
# * 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(48) @ 2: uvm_test_top [run_phase] RST is deasserted
# UVM_INFO FIFO_test.sv(50) @ 2: uvm_test_top [run_phase] stimulus generation write_only_seq started
# UVM_INFO FIFO_test.sv(52) @ 22: uvm_test_top [run_phase] stimulus generation write_only_seq ended
# UVM_INFO FIFO_test.sv(55) @ 22: uvm_test_top [run_phase] stimulus generation read_only_seq started
# UVM_INFO FIFO_test.sv(57) @ 42: uvm_test_top [run_phase] stimulus generation read_only_seq ended
# UVM_INFO FIFO_test.sv(59) @ 42: uvm_test_top [run_phase] stimulus generation write_read_seq started
# UVM_INFO FIFO_test.sv(61) @ 2042: uvm_test_top [run_phase] stimulus generation write_read_seq ended
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 2042: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO FIFO_score.sv(102) @ 2042: uvm_test_top.env.sb [report_phase] Total successful Transaction: 1021
# UVM_INFO FIFO_score.sv(103) @ 2042: uvm_test_top.env.sb [report_phase] Total Failed Transaction: 0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   14
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# [report_phase]     2
# [run_phase]     8
# ** Note: $finish    : C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 2042 ns  Iteration: 61  Instance: /top
```

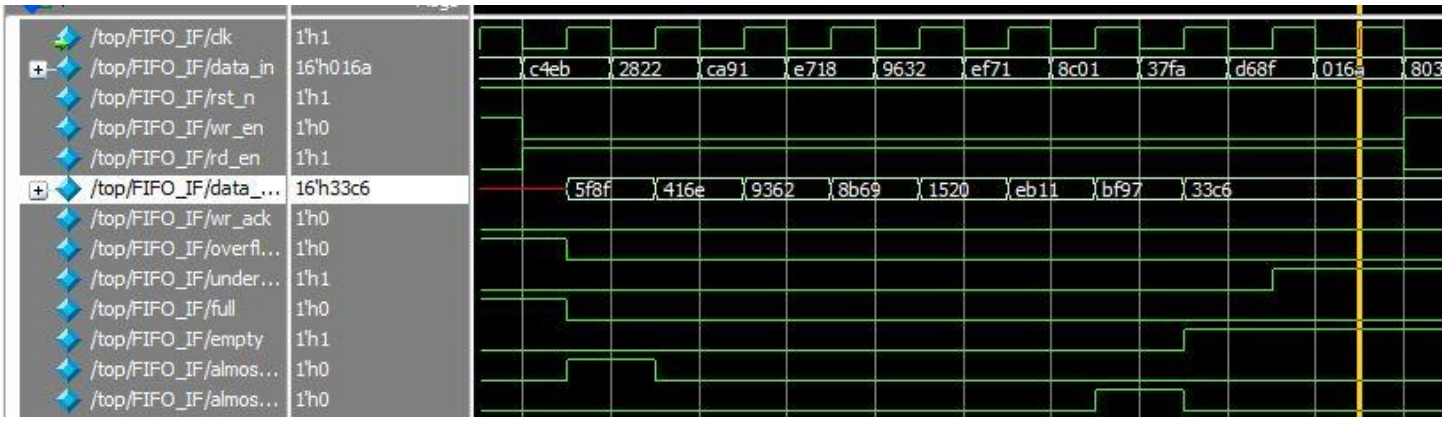
23- Wave

Write only sequence

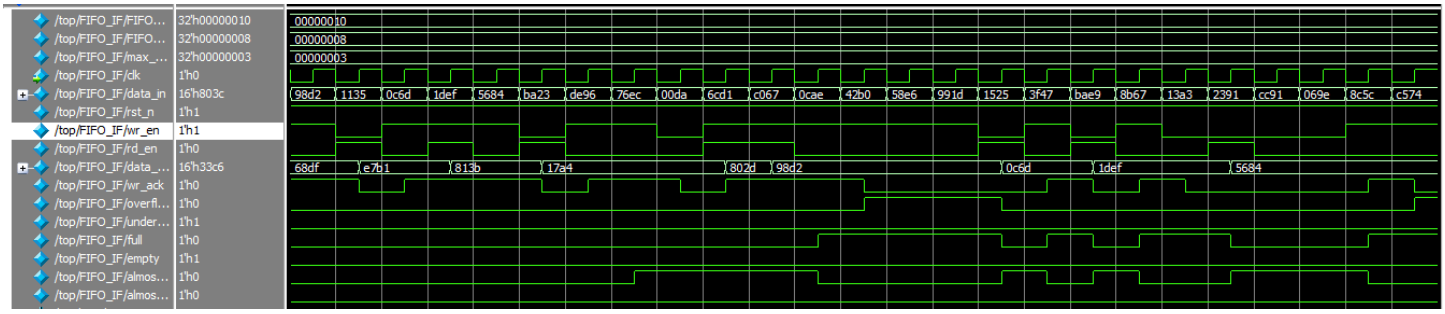
Dataout is don’t care because no read operation yet



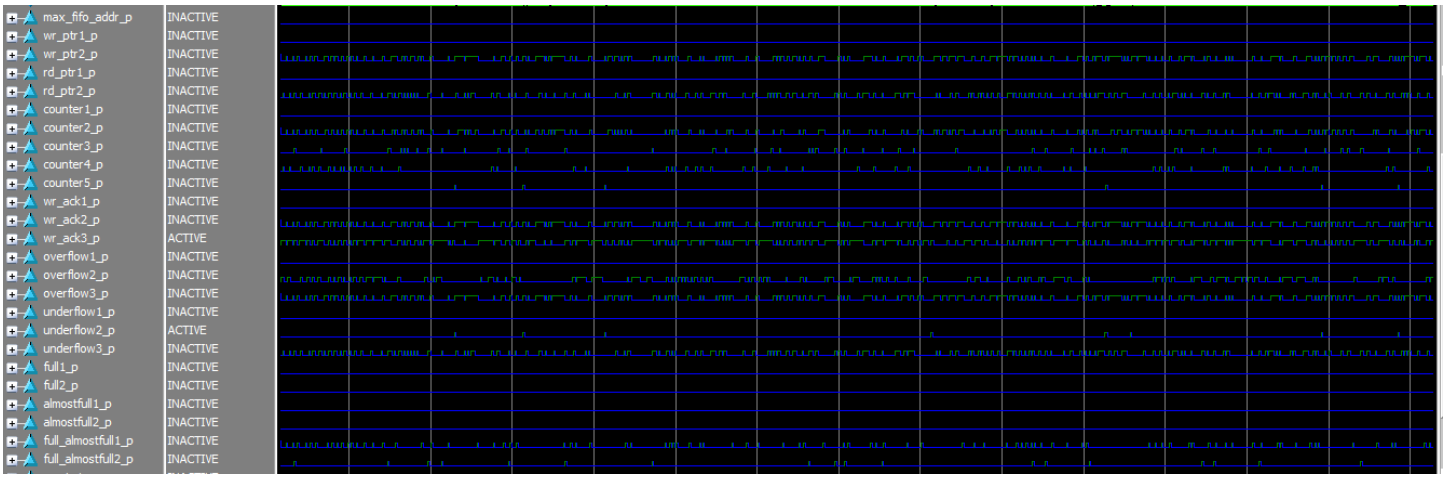
Read only sequence



Write Read sequence



Assertions


max_fifo_addr_p INACTIVE
wr_ptr1_p INACTIVE
wr_ptr2_p INACTIVE
rd_ptr1_p INACTIVE
rd_ptr2_p INACTIVE
counter1_p INACTIVE
counter2_p INACTIVE
counter3_p INACTIVE
counter4_p INACTIVE
counter5_p INACTIVE
wr_ack1_p INACTIVE
wr_ack2_p INACTIVE
wr_ack3_p ACTIVE
overflow1_p INACTIVE
overflow2_p INACTIVE
overflow3_p INACTIVE
underflow1_p INACTIVE
underflow2_p ACTIVE
underflow3_p INACTIVE
full1_p INACTIVE
full2_p INACTIVE
almostfull1_p INACTIVE
almostfull2_p INACTIVE
full_almostfull1_p INACTIVE
full_almostfull2_p INACTIVE


empty1_p INACTIVE
empty2_p INACTIVE
almostempty1_p INACTIVE
almostempty2_p INACTIVE
empty_almostempt... INACTIVE
empty_almostempt... INACTIVE