UVM project - Synchronous FIFO

Abdelrahman Mohamed Ali

7 May, 2024

Gmail	**abdelrahmansalby23@gmail.com**
LinkedIn	**Abdelrahman Mohamed**
GitHub	**Abdelrahman1810**

Please see readme.md file first
Note: if you use VS code use shortcut ‘ ctrl+shift+v ‘ so you can see the file with GUI

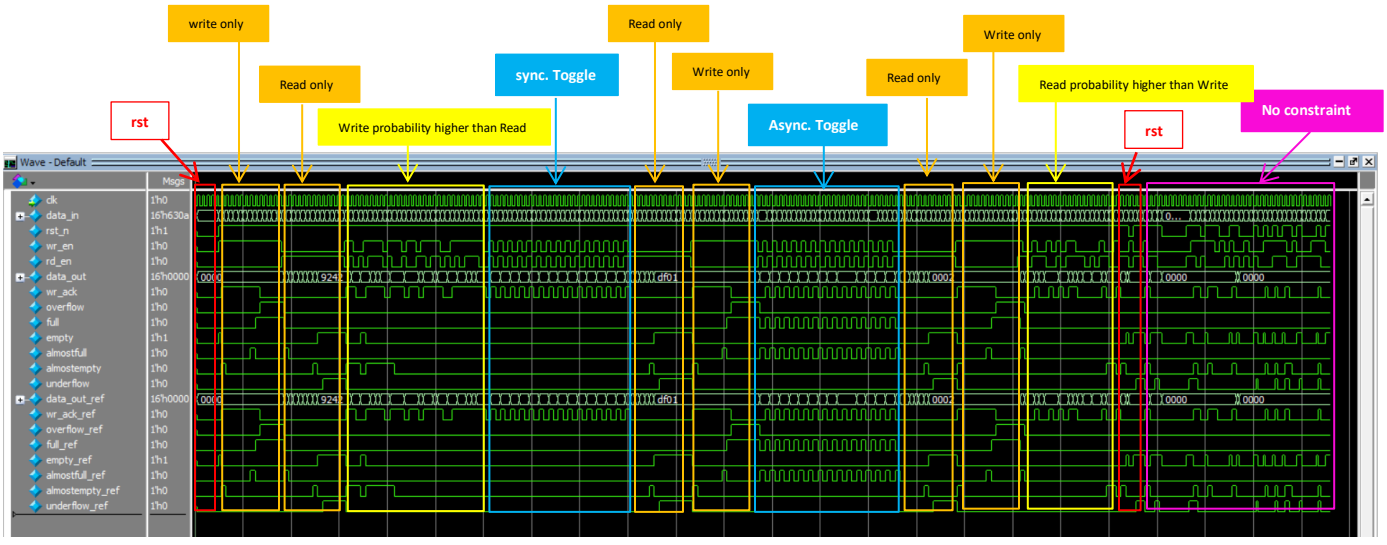
Verification Plane

1. Activate reset at the first 5 cycle.
2. LOCK RESET
3. Write only to make FIFO full.
4. Write when FIFO full.
5. Read times less than FIFO_DEPTH.
6. read and write constraint `WRITE DIST MORE THAN READ`.
    - the first loop to make data_in constraint one bit high
    - the seconed loop with no constraint at data_in
7. Apply Sync toggle => read and write will be equal but they are not equal to the same value twice in a row.
8. Read only to make FIFO empty.
9. Read when FIFO empty.
10. Write only to make FIFO before full.
11. Apply Async toggle => read and write will be not equal but they are not equal to the same value twice in a row.
12. Read Only.
13. write Only.
14. UNLOCK RESET
15. Apply read and write at the same time `WRITE DIST LESS THAN READ`.
    - the first loop to make data_in constraint one bit high
    - the seconed loop with no constraint at data_in
16. Activate reset.
17. Randomization with no Constraint.

Label	Description	Stimulus Generation	Functional Coverage	Functionality Check
FIFO_1	asserted the reset at the first 5 clk cycle	Randomization with post_randomize dunction to make sure that at the first 5 clk cycle the reset will be low	included in covergroup CVG in coverpoint of reset named "rst_cp"	_____
FIFO_2	Case: reset activated Check the sequential signal (wr_ack, overflow, underflow, data_out) should be zero	Randomization with post_randomize dunction to make sure that at the first 5 clk cycle the reset will be low	included as a coverpoint of reset and wr_ack, overflow, underflow, data_out included as a cross coverage to check that signal will be in active when reset is active	Output Checked against golden model and checked with combinational assertion (label: assert_wr_akc_rst assert_overflow_rst assert_underflow_rst assert_data_out_rst)
FIFO_3	Case: reset is activated the internal signal (wr_otr, rd_ptr and count) should be inactive	Randomization with post_randomize dunction to make sure that at the first 5 clk cycle the reset will be low	included as a coverpoint of rst_n No coverage for internal signal	Output checked with assertion (label: assert_count_rst, assert_wr_ptr_rst, assert_rd_ptr_rst)
FIFO_4	make reset Inactive most of the time	Randomization under constraint (label: CON_RESET) to make rst_n is active => 95%, and rst_n is Inactive => 5%	included in covergroup CVG in coverpoint of reset named "rst_cp"	_____
FIFO_5	Case: wr_en = 1, rd_en = 0, full = 0, empty = 0. 1.Write operation done 2.wr_ack gets high 3.if count ==FIFO_DEPTH-1 4.then almostfull gets high	Randomization under constraint (label: CON _W) that constraint wr_en to be active and rd_en to be inactive	Included a coverpoint wr_ack_cp. Included a coverpoint almostfull_cp. Included a cross almostfull & wr_en. Included a cross coverage wr_ack & wr_en.	Output Checked against golden model Checked by assertion Label: **ack_active,** **almostfull_count**
FIFO_6	Case: wr_en = 1, rd_en = 0, full = 1. 1.Write operation ignored 2.wr_ack gets low 3.overflow gets high	Randomization under constraint (label: CON _W) that constraint wr_en to be active and rd_en to be inactive	Included a coverpoint overflow_cp Included a cross coverage overflow & wr_en.	Output Checked against golden model Checked by assertion label: **ack_inactive** **overflow_active**
FIFO_7	Case: wr_en = 1, rd_en = 0, empty = 1. 1.Write operation done 2.wr_ack gets high 3.empty gets low 4.almostempty gets high	Randomization under constraint (label: CON _W) that constraint wr_en to be active and rd_en to be inactive	Included a coverpoint wr_ack_cp. Included a coverpoint for empty. Included a coverpoint almostempty_cp. Included a cross coverage wr_ack & empty (ack_empty_cross) Included a cross coverage almostempty & empty (almostempty_empty_cross)	Output Checked against golden model Checked by assertion Label: **ack_active,** **empty_inactive,** **almostempty_from_empty**
FIFO_8	Case: wr_en = 1, rd_en = 0, almostfull = 1. 1.Write operation done 2.wr_ack gets high 3.almostfull gets low 4.count == FIFO_DEPTH so full gets high	Randomization under constraint (label: CON _W) that constraint wr_en to be active and rd_en to be inactive	Included a coverpoint wr_ack_cp. Included a coverpoint full_cp. Included a coverpoint almostfull_cp. Included a cross coverage wr_ack & full (ack_full_wr_cross) Included a cross coverage almostfull & full (almostfull_full_cross)	Output Checked against golden model Checked by assertion Label: **ack_inactive,** **almostempty_inactive,** **full_count**
FIFO_9	Case: wr_en = 1, rd_en = 0, almostempty = 1. 1.Write operation done 2.wr_ack gets high almostempty gets low	Randomization under constraint (label: CON _W) that constraint wr_en to be active and rd_en to be inactive	Included a coverpoint wr_ack_cp. Included a coverpoint almostempty_cp. Included a cross coverage wr_ack & empty (ack_empty_cross) Included a cross coverage almostempty & wr_ack (ack_almostempty_cross)	Output Checked against golden model Checked by assertion Label: **almostempty_inactive**
FIFO_10	Case: wr_en = 0, rd_en = 1, empty = 0, full = 0. 1.Read operation done 2.check data_out	Randomization under constraint (label: CON _R) that constraint wr_en to be inactive and rd_en to be active	Included a coverpoint data_out_cp. Included a cross coverage data_out &rd_en	Output Checked against golden model Checked by assertion Label: **almostempty_count**
FIFO_11	Case: wr_en = 0, rd_en = 1, empty = 1. 1.Read operation ignored 2.Underflow gets high	Randomization under constraint (label: CON _R) that constraint wr_en to be inactive and rd_en to be active	Included a coverpoint underflow_cp Included a cross coverage: underflow &rd_en, underflow &empty	Output Checked against golden model Checked by assertion Label: **underflow_active**
FIFO_12	Case: wr_en = 0, rd_en = 1, full = 1. 1.Read operation done 2.Check data_out 3.full gets low 4.almostfull gets high	Randomization under constraint (label: CON _R) that constraint wr_en to be inactive and rd_en to be active	Included a coverpoint data_out_cp. Included a coverpoint full_cp. Included a coverpoint almostfull_cp. Included a cross coverage almostfull & full (almostfull_full_cross)	Output Checked against golden model Checked by assertion Label: **almostfull_from_full**
FIFO_13	Case: wr_en = 0, rd_en = 1, almostempty = 1. 1.Read operation done 2.Checked data_out 3.almostempty gets low 4.empty gets high	Randomization under constraint (label: CON _R) that constraint wr_en to be inactive and rd_en to be active	Included a coverpoint data_out_cp. Included a coverpoint emoty_cp. Included a coverpoint almostempty_cp. Included a cross coverage almostempty & empty (ack_empty_cross)	Output Checked against golden model Checked by assertion Label: **empty_from_almost**
FIFO_14	Case: wr_en=0, rd_en = 1, almostfull = 1. 1.Read operation done 2.Checked data_out 3.almostfull gets low	Randomization under constraint (label: CON _R) that constraint wr_en to be inactive and rd_en to be active	Included a coverpoint almostfull_cp. Included a cross coverage almostfull &rd_en (almostfull_cross)	Output Checked against golden model Checked by assertion Label: **almostfull_inactive**
FIFO_15	Case: wr_en=1, rd_en=1, full = 0, empty = 0. 1.write operation done 2.wr_ack gets high 3.read operation done	Randomization under constraint (label: CON_BOTH_ACTIVE) that constraint wr_en to be inactive and rd_en to be active	Included a coverpoint wr_ack_cp Included a coverpoint data_out_cp Included a cross coverage: wr_ack & wr_en (ack_wr_rd_cross).	Output Checked against golden model Checked by assertion Label: **ack_active,** **count_noChange**

	4.checked data_out 5.Count will not change		data_out&wr_en&rd_en (data_out_cross).	
FIFO_16	Case: wr_en=1, rd_en=1, full = 1. 1.write operation ignored 2.wr_ack gets low 3.read operation done 4.checked data_out 5.full gets low 6.almostfull gets high 7.overflow gets high 8.count will decrement.	Randomization under constraint (label: CON_BOTH_ACTIVE) that constraint wr_en to be inactive and rd_en to be active	Included a coverpoint data_out_cp. Included a coverpoint full_cp. Included a coverpoint almostfull_cp. Included a coverpoint overflow_cp. Included a cross coverage: data_out&wr_en&rd_en (data_out_cross). almostfull & full (almostfull_full_cross) overflow& full (full_overflow_cross)	Output Checked against golden model Checked by assertion Label: **ack_inactive,** **full_inactive,** **almostfull_from_full,** **overflow_active,** **count_dec**
FIFO_17	Case: wr_en=1, rd_en=1, empty = 1. 1.write operation done 2.wr_ack gets high 3.read operation ignored 4.empty gets low 5.almostempty gets high 6.underflow gets high 7.count will increment.	Randomization under constraint (label: CON_BOTH_ACTIVE) that constraint wr_en to be inactive and rd_en to be active	Included a coverpoint data_out_cp. Included a coverpoint wr_ack_cp. Included a coverpoint emoty_cp. Included a coverpoint almostempty_cp. Included a coverpoint underflow_cp. Included a cross coverage: data_out&wr_en&rd_en (data_out_cross). underflow & emoty (empty_underflow_cross) emoty & almostempty (almostempty_empty_cross) wr_ack&wr_en&rd_en (ack_wr_rd_cross)	Output Checked against golden model Checked by assertion Label: **ack_active,** **empty_inactive,** **almostempty_from_empty,** **underflow_active,** **count_inc**
FIFO_18	Case: wr_en=1, rd_en=1, almostfull = 1. 1.write operation done 2.wr_ack gets high 3.read operation done 4.checked data_out 5. almostfull remain high 6.count remain the same.	Randomization under constraint (label: CON_BOTH_ACTIVE) that constraint wr_en to be inactive and rd_en to be active	Included a coverpoint data_out_cp. Included a coverpoint wr_ack_cp. Included a cross coverage: data_out&wr_en&rd_en (data_out_cross). wr_ack&wr_en&rd_en (ack_wr_rd_cross)	Output Checked against golden model Checked by assertion Label: **ack_active,** **almostfull_noChange,** **count_noChange**
FIFO_19	Case: wr_en=1, rd_en=1, almostempty = 1. 1.write operation done 2.wr_ack gets high 3.read operation done 4.checked data_out 5. almostempty remain high 6.count remain the same.	Randomization under constraint (label: CON_BOTH_ACTIVE) that constraint wr_en to be inactive and rd_en to be active	Included a coverpoint data_out_cp. Included a coverpoint wr_ack_cp. Included a cross coverage: data_out&wr_en&rd_en (data_out_cross). wr_ack&wr_en&rd_en (ack_wr_rd_cross)	Output Checked against golden model Checked by assertion Label: **ack_active,** **almostempty_noChange,** **count_noChange**

Verification flow in waveform



UVM FLOW

1. create defnitions.svh contain all defnithion macro i have used in the program
2. creat shared_pkg.sv that have shared variable and sequence for assertion
3. creat interface
4. add FIFO.sv `design file` in DUT folder
5. creat a golden model to check the DUT output
6. creat an assertion file
7. create top.sv file
    - pass the data from interface to DUT and refrence model
    - bind file assertion to design
    - set the virtual interface to data base
    - run UVM test
8. create configration class so can get virtul interface by
9. create sequence_item
    - add the variable and rand variable
    - creat tow methods {print_DUT(), print_REF()}
    - creat a constraint block
10. create 9 FIFO_sequence so can verify the design
    - write_only_sequence
    - read_only_sequence
    - write_read_sequence
    - sync_toggle_sequence
    - Async_toggle_sequence
    - rst_sequence
    - random_sequence
11. create a driver.sv file
    - create avirtual interface between driver and real interface
    - make driver a producer get_data
    - assigned data from sequenceItem and pass it to interface `inputs`
12. create a monitor.sv file
    - create avirtual interface between monitor and real interface
    - assigned data from virtual interface and pass it to monitor object from sequenceItem `inputs and outputs`
    - create analysis_port to send data to agent
13. create a sequencer
14. create an agent.sv
    - get confegration object from data base and pass it to monitor and driver
    - make connection between monitor and agent to send data to scoreboard and cover_collector
    - make connection between sequencer and driver
15. create env.sv
    - create component {agent, scoreboard, cover_collector}
    - connect between agent and {scoreboard, cover_collector}
16. create test.sv file
    - get virtual interface an pass it to env
    - creat object from sequences
    - call bult-in function raise_objection to indecate that test is start
    - run sequences object in run_phase with specific sequence I have explaned in pdf
    - creat component for env
    - call bult-in function drop_objection to indecate that test is finished

/// Do file

```
vlib work
vlog -coveropt 3 +cover +acc {shared_pkg\shared_pkg.sv}
###
vlog -coveropt 3 +cover +acc {Interface\interface.sv}
vlog -coveropt 3 +cover +acc {refrence\FIFO_ref.sv}
###
vlog -coveropt 3 +cover +acc {DUT\FIFO.sv}
vlog +define+LOOK_ASSERTION -coveropt 3 +cover +acc {DUT\assertion.sv}
###
vlog -coveropt 3 +cover +acc {objects\configration.sv}

#vlog -coveropt 3 +cover +acc {objects\sequence_Items\sequenceItem_Valid.sv}
vlog -coveropt 3 +cover +acc {objects\sequence_Item\sequenceItem.sv}

vlog -coveropt 3 +cover +acc {objects\FIFO_sequence\FIFO_reset_sequence.sv}
vlog -coveropt 3 +cover +acc {objects\FIFO_sequence\FIFO_write_only_sequence.sv}
vlog -coveropt 3 +cover +acc {objects\FIFO_sequence\FIFO_read_only_sequence.sv}
vlog -coveropt 3 +cover +acc {objects\FIFO_sequence\FIFO_write_read_sequence.sv}
vlog -coveropt 3 +cover +acc {objects\FIFO_sequence\FIFO_sync_toggle_sequence.sv}
vlog -coveropt 3 +cover +acc {objects\FIFO_sequence\FIFO_Async_toggle_sequence.sv}
vlog -coveropt 3 +cover +acc {objects\FIFO_sequence\FIFO_random_sequence.sv}
###
vlog -coveropt 3 +cover +acc {UVM_top\Test\env\agent\driver\driver.sv}
vlog -coveropt 3 +cover +acc {UVM_top\Test\env\agent\monitor\monitor.sv}
vlog -coveropt 3 +cover +acc {UVM_top\Test\env\agent\sequencer\sequencer.sv}
vlog -coveropt 3 +cover +acc {UVM_top\Test\env\agent\agent.sv}
###
vlog -coveropt 3 +cover +acc {UVM_top\Test\env\coverage_collector\coverage_collector.sv}
vlog -coveropt 3 +cover +acc {UVM_top\Test\env\scoreboard\scoreboard.sv}
vlog -coveropt 3 +cover +acc {UVM_top\Test\env\env.sv}
###
vlog -coveropt 3 +cover +acc {UVM_top\Test\test.sv}
vlog -coveropt 3 +cover +acc {UVM_top\top.sv}

#vsim -voptargs=+acc work.top -classdebug -uvmcontrol=all
vsim +UVM_VERBOSITY=UVM_LOW -voptargs=+acc work.top -classdebug -uvmcontrol=all -cover

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

run -all
#quit -sim
```

/// Interface

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

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

logic [FIFO_WIDTH-1:0] data_out_ref;
logic wr_ack_ref, overflow_ref;
logic full_ref, empty_ref, almostfull_ref, almostempty_ref, underflow_ref;
endinterface //FIRO_interface
```

/// shared pkg

```
package shared_pkg;
`include "defines/defines.svh"
parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;
parameter RANGE = (2**FIFO_WIDTH) - 1;
parameter ACTIVE = 1;
parameter INACTIVE = 0;

int WR_EN_ON_DIST = 70;
int RD_EN_ON_DIST = 30;
int RESET_ACTIVE = 5;
int WR_START_VALUE_OF_TOGGLE = 1;

parameter ZERO = 0;
parameter MAX = (2**FIFO_DEPTH) - 1;
reg [FIFO_WIDTH-1:0] one_bit_high [16] = '{16'h1, 16'h2, 16'h4, 16'h8, 16'h10, 16'h20, 16'h40, 16'h80,
                                           16'h100, 16'h200, 16'h400, 16'h800, 16'h1000, 16'h2000, 16'h4000, 16'h8000};
int error_counter = 0;
int correct_counter = 0;
int inc_correct_counter = 0;

parameter DEPTH_LOOP = FIFO_DEPTH * 2;
parameter BEFORE_ALMOST = FIFO_DEPTH - 2;
parameter HOBBIT_LOOP = 5_00;
parameter DWARF_LOOP = 10_00;
parameter HUMAN_LOOP = 20_00;
parameter GIANT_LOOP = 30_00;
parameter BOOM_LOOP = 50_00;

/*
isFIRST_DIST_FINITH
this parameter as a flag: when sequence FIFO_write_read_sequence call for the first time in uvm_test
will make WRITE active more probability than READ then isFIRST_DIST_FINITH get high for the seconed time
uvm_test call sequence FIFO_write_read_sequence will make READ active more probability than WRITE
*/
bit isFIRST_DIST_FINITH = 0;
bit isFirstRead = 0;
bit isWriteTOfull = 0;

// Sequences
sequence A_A(Active1, Active2);
  (Active1 && Active2);
endsequence

sequence A_I(Active, Inactive);
  (Active && !Inactive);
endsequence

sequence AA_I(Active1, Active2, Inactive);
  (Active1 && Active2 && !Inactive);
endsequence

sequence A_II(Active, Inactive1, Inactive2);
  (Active && !Inactive1 && !Inactive2);
endsequence
endpackage
```

/// FIFO_ref.sv

```
import shared_pkg::*;
module FIFO_ref(data_in, wr_en, rd_en, clk, rst_n, full, empty, almostfull, almostempty, wr_ack, overflow, underflow, data_out);
    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, underflow;
    output full, empty, almostfull, almostempty;
    reg [FIFO_WIDTH-1:0] fifoo_q [$];
    // Write
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            fifoo_q.delete();
            wr_ack <= 0;
            overflow <= 0;
        end
        else if (wr_en && !full) begin
            fifoo_q.push_back(data_in);
            wr_ack <= 1;
            overflow <= 0;
        end
        else begin
            wr_ack <= 0;
            if (full && wr_en)
                overflow <= 1;
            else
                overflow <= 0;
        end
    end
    // Read
```

```
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            fifoo_q.delete();
            underflow <= 0;
            data_out <= 0;
        end
        else if (rd_en && !empty) begin
            data_out <= fifoo_q.pop_front();
            underflow <= 0;
        end
        else begin
            if (empty && rd_en)
                underflow <= 1;
            else
                underflow <= 0;
        end
    end

    assign almostfull = (fifoo_q.size() == FIFO_DEPTH-1 && rst_n)? 1:0;
    assign full = (fifoo_q.size() >= FIFO_DEPTH && rst_n)? 1:0;

    assign empty = (fifoo_q.size() == 0 && rst_n)? 1:0;
    assign almostempty = (fifoo_q.size() == 1 && rst_n)? 1:0;
endmodule
```

/// FIFO.sv

```
import shared_pkg::*;
`include "defines/defines.svh"
module FIFO(data_in, wr_en, rd_en, clk, rst_n, full, empty, almostfull, almostempty, wr_ack, overflow, underflow, data_out);

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, underflow;
output full, empty, almostfull, almostempty;
//clk, rst_n, wr_en, rd_en, data_in, data_out,
//wr_ack, overflow, 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;
        wr_ack <= 0; // FIX
        overflow <= 0; // FIX
    end
    else if (wr_en && count < FIFO_DEPTH) begin
        mem[wr_ptr] <= data_in;
        wr_ack <= 1;
        wr_ptr <= wr_ptr + 1;
        overflow <= 0; // FIX
    end
    else begin
        wr_ack <= 0;
        if (full && wr_en)// FIX
            overflow <= 1;
        else
            overflow <= 0;
    end
end

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

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        count <= 0;
    end
    else begin
        if (({wr_en, rd_en} == 2'b11) && full) // FIX
            count <= count - 1; // FIX
```

```
        else if (({wr_en, rd_en} == 2'b11) && empty) // FIX
            count <= count + 1; // FIX
        else 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 && rst_n)? 1 : 0; // FIX

assign almostfull = (count == FIFO_DEPTH-1)? 1 : 0;
assign almostempty = (count == 1)? 1 : 0;

`ifdef LOOK_ASSERTION
sequence inc_ptr(ptr);
    ($past(ptr)+1 == ptr)||($past(ptr)==FIFO_DEPTH-1);
/*
($past(ptr)==FIFO_DEPTH-1) => becouse for questa when ptr.past = 7 then ptr should be zero but questa will treats it as 8 NOT zero
when $past(ptr) = 7 then $past(ptr) + 1 = 0 (3-bit)
for questa $past(ptr) + 1 = 8
*/
endsequence

// Internal  signal assertion
// Count assertion
count_inc: `asrt_prp(A_II(wr_en, rd_en, full) |=> ($past(count)+1 == count));
count_dec: `asrt_prp(A_II(rd_en, wr_en, empty)|=> ($past(count)-1 == count));
count_noChange: `asrt_prp(`same_seq |=> ($past(count) == count));
// Count Cover
count_inc_cover:      `cov_prp(A_II(wr_en, rd_en, full) |=> ($past(count)+1 == count));
count_dec_cover:      `cov_prp(A_II(rd_en, wr_en, empty)|=> ($past(count)-1 == count));
count_noChange_cover: `cov_prp(`same_seq |=> ($past(count) == count));

// pointer assertion
inc_wr_ptr_assert:  `asrt_prp(A_A(wr_en,(count < FIFO_DEPTH)) |=> inc_ptr(wr_ptr) );
inc_rd_ptr_assert:  `asrt_prp((rd_en && count != 0) |=> inc_ptr(rd_ptr) );
// pointer cover
inc_wr_ptr_cover:  `cov_prp( A_A(wr_en,(count < FIFO_DEPTH)) |=> inc_ptr(wr_ptr) );
inc_rd_ptr_cover:  `cov_prp((rd_en && count != 0) |=> inc_ptr(rd_ptr) );

// reset assertion and cover
always_comb begin : rst_n_assert
  if (!rst_n) begin
    assert_count_rst:  `asrt_fn(count == 0);
    assert_wr_ptr_rst: `asrt_fn(wr_ptr == 0);
    assert_rd_ptr_rst: `asrt_fn(rd_ptr == 0);

    cover_count_rst:  `cov_fn(count == 0);
    cover_wr_ptr_rst: `cov_fn(wr_ptr == 0);
    cover_rd_ptr_rst: `cov_fn(rd_ptr == 0);
  end
end

// assertion of signals related with count
full_count:         `asrt_prp((count >= FIFO_DEPTH) |->  full);
almostfull_count:       `asrt_prp((count==FIFO_DEPTH-1) |-> almostfull);
almostempty_count:     `asrt_prp((count==1) |-> almostempty );
empty_count:         `asrt_prp((count==ZERO) |-> empty );
// cover of signals related with count
full_count_cover:       `cov_prp((count >= FIFO_DEPTH) |->  full);
almostfull_count_cover:     `cov_prp((count==FIFO_DEPTH-1) |-> almostfull);
almostempty_count_cover:        `cov_prp((count==1) |-> almostempty );
empty_count_cover:       `cov_prp((count==ZERO) |-> empty );

`endif // LOOK_ASSERTION
endmodule
```

/// assertion.sv

```
import shared_pkg::*;
`include "defines/defines.svh"

module FIFO_sva(data_in, wr_en, rd_en, clk, rst_n, full, empty, almostfull, almostempty, wr_ack, overflow, underflow, data_out);

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

`ifdef LOOK_ASSERTION
////////////////////////////
//        Assertion       //
////////////////////////////
always_comb begin : rst_n_assert
  if (!rst_n) begin
    assert_wr_akc_rst:    `asrt_fn(wr_ack == 0);
    assert_overflow_rst:  `asrt_fn(overflow == 0);
    assert_underflow_rst: `asrt_fn(underflow == 0);
    assert_data_out_rst:  `asrt_fn(data_out == 0);
```

```
    assert_full_rst:        `asrt_fn(full == 0);
    assert_almostfull_rst:  `asrt_fn(almostfull == 0);
    assert_empty_rst:       `asrt_fn(empty == 0);
    assert_almostempty_rst: `asrt_fn(almostempty == 0);
  end
end

// Global signal assertion
// full
full_from_almost: `asrt_prp(AA_I(almostfull,wr_en,rd_en) |=> full);
full_noChange:    `asrt_prp( `same_seq |=> $past(full) == full);
full_inactive:    `asrt_prp(A_A(full,rd_en) |=> !full);

// wr_ack
ack_active:   `asrt_prp(A_I(wr_en,full) |=> wr_ack);
ack_inactive: `asrt_prp( full |=> !wr_ack);

// almostfull
almostfull_from_full: `asrt_prp(A_A(full,rd_en)  |=> ($fell(full) && $rose(almostfull)));
almostfull_noChange:  `asrt_prp( `same_seq |=> $past(almostfull) == almostfull);
almostfull_inactive:  `asrt_prp(AA_I(almostfull,rd_en,wr_en) |=> !almostfull );

// overflow
overflow_active:`asrt_prp(A_A(full,wr_en) |=> overflow);
overflow_wr_in: `asrt_prp(($past(overflow) && !wr_en) |=> !overflow);
overflow_Nfull: `asrt_prp(($past(overflow) && !full)  |=> !overflow);

// almostempty
almostempty_noChange:  `asrt_prp( `same_seq |=> ($past(almostempty) == almostempty));
almostempty_from_empty:`asrt_prp(A_A(empty,wr_en) |=> ($fell(empty) && $rose(almostempty)));
almostempty_inactive:  `asrt_prp(AA_I(almostempty,wr_en,rd_en) |=> !almostempty );

// empty
empty_noChaneg:    `asrt_prp( `same_seq |=> ($past(empty) == empty) );
empty_from_almost: `asrt_prp(AA_I(almostempty,rd_en,wr_en) |=> empty);
empty_inactive:    `asrt_prp(A_A(empty,wr_en) |=> !empty);

// underflow
underflow_active:  `asrt_prp(A_A(empty,rd_en) |=> underflow);
underflow_Nempty:  `asrt_prp(($past(underflow) && !empty)  |=> !underflow);
underflow_Nrd:     `asrt_prp(($past(underflow) && !rd_en)  |=> !underflow);

//////////////////////////
//        Coverage      //
//////////////////////////
always_comb begin : rst_n_cover
  if (!rst_n) begin
    cover_wr_akc_rst:    `cov_fn(wr_ack == 0);
    cover_overflow_rst:  `cov_fn(overflow == 0);
    cover_underflow_rst: `cov_fn(underflow == 0);
    cover_data_out_rst:  `cov_fn(data_out == 0);



    cover_full_rst:        `cov_fn(full == 0);
    cover_almostfull_rst:  `cov_fn(almostfull == 0);
    cover_empty_rst:       `cov_fn(empty == 0);
    cover_almostempty_rst: `cov_fn(almostempty == 0);
  end
end

// Global signal cover
// full
full_from_almost_cover: `cov_prp(AA_I(almostfull,wr_en,rd_en) |=> full);
full_noChange_cover:    `cov_prp( `same_seq |=> !full);
full_inactive_cover:    `cov_prp(A_A(full,rd_en) |=> !full);

// wr_ack
ack_active_cover:   `cov_prp(A_I(wr_en,full) |=> wr_ack);
ack_inactive_cover: `cov_prp(A_A(full,wr_en)  |=> !wr_ack);

// almostfull
almostfull_from_full_cover: `cov_prp(A_A(full,rd_en)  |=> ($fell(full) && $rose(almostfull)));
almostfull_noChange_cover:  `cov_prp( `same_seq |=> $past(almostfull) == almostfull);
almostfull_inactive_cover:  `cov_prp(AA_I(almostfull,rd_en,wr_en) |=> !almostfull );

// overflow
overflow_active_cover:`cov_prp(A_A(full,wr_en) |=> overflow);
overflow_wr_in_cover: `cov_prp(($past(overflow) && !wr_en) |=> !overflow);
overflow_Nfull_cover: `cov_prp(($past(overflow) && !full)  |=> !overflow);

// almostempty
almostempty_noChange_cover:   `cov_prp( `same_seq |=> ($past(almostempty) == almostempty));
almostempty_from_empty_cover: `cov_prp(A_A(empty,wr_en) |=> ($fell(empty) && $rose(almostempty)));
almostempty_inactive_cover:   `cov_prp(AA_I(almostempty,wr_en,rd_en) |=> !almostempty );

// empty
empty_noChaneg_cover:    `cov_prp( `same_seq |=> ($past(empty) == empty) );
empty_from_almost_cover: `cov_prp(AA_I(almostempty,rd_en,wr_en) |=> empty);
empty_inactive_cover:    `cov_prp(A_A(empty,wr_en) |=> !empty);

// underflow
underflow_active_cover:   `cov_prp(A_A(empty,rd_en) |=> underflow);
underflow_Nempty_cover:   `cov_prp(($past(underflow) && !empty)  |=> !underflow);
underflow_Nrd_cover:      `cov_prp(($past(underflow) && !rd_en)  |=> !underflow);
`endif // LOOK_ASSERTION
```

```
endmodule
```

/// sequenceItem.sv

```
package sequenceItem_pkg;
import shared_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
`include "defines/defines.svh"

// Sequence Item class Valid and Invalid
class FIFO_sequenceItem extends uvm_sequence_item;
    `uvm_object_utils(FIFO_sequenceItem)
// input
    rand bit [FIFO_WIDTH-1:0] data_in;
    rand bit rst_n, wr_en, rd_en;

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

// output refrence
    bit [FIFO_WIDTH-1:0] data_out_ref;
    bit wr_ack_ref, overflow_ref;
    bit full_ref, empty_ref, almostfull_ref, almostempty_ref, underflow_ref;

// new
    `Func_new("FIFO_sequenceItem")



// my function to print FIFO DUT and REF
    function string print_DUT();
        return $sformatf("DUT:\ndata_out = %0d, wr_ack = %0d, full = %0d, empty = %0d, underflow = %0d, almostempty = %0d, almostfull = %0d, overflow =
%0d",
        data_out, wr_ack, full, empty, underflow, almostempty, almostfull, overflow
        );
    endfunction

    function string print_REF();
        return $sformatf("REF:\ndata_out_ref = %0d, wr_ack_ref = %0d, full_ref = %0d, empty_ref = %0d, underflow_ref = %0d, almostempty_ref = %0d,
almostfull_ref = %0d, overflow_ref = %0d",
        data_out_ref, wr_ack_ref, full_ref, empty_ref, underflow_ref, almostempty_ref, almostfull_ref, overflow_ref
        );
    endfunction

////// ////// ////// ////// //////
//       Constraint block        //
////// ////// ///// ////// //////
// RESET CONSTRAINT
    constraint CON_RESET {
        rst_n dist {1:=100-RESET_ACTIVE, 0:=RESET_ACTIVE};
    }
// WRITE & READ CONSTRAINT ** DIST PROBABILITY **
    constraint CON_W_R {
        wr_en dist {ACTIVE:=WR_EN_ON_DIST, INACTIVE:=100-WR_EN_ON_DIST};
        rd_en dist {ACTIVE:=RD_EN_ON_DIST, INACTIVE:=100-RD_EN_ON_DIST};
    }

// ONLY WRITE CONSTRAINT
    constraint CON_W {
        wr_en == ACTIVE;
        rd_en == INACTIVE;
    }

// ONLY READ CONSTRAINT
    constraint CON_R {
        wr_en == INACTIVE;
        rd_en == ACTIVE;
    }

// CONSTRAINT TO MAKE READ AND WRITE ALWAYS HIGH
    constraint CON_BOTH_ACTIVE {
        wr_en == ACTIVE;
        rd_en == ACTIVE;
    }

// CONSTRAINT DATA_IN = ONE_BIT_HIGH
    constraint CON_DATA_ONE_BIT{
        $countones(data_in) == 1;
    }

// CONSTRAINT DATA_IN = MAX || ZERO || OTHER_VALUE
    constraint CON_DATA_MAX_ZERO{
        data_in dist {MAX:=5, ZERO:=5, [0:RANGE]:/90};
    }
////// ////// ////// ////// ////// //////
//       Constraint block finish        //
////// ////// ///// ////// ////// //////

endclass //FIFO_sequenceItem extends uvm_sequence_item
endpackage
```

/// configration.sv

```
`timescale 1ps/1ps
package config_pkg;
import shared_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
`include "defines/defines.svh"

// Configration class
class FIFO_config extends uvm_object;
    `uvm_object_utils(FIFO_config)
    virtual FIFO_interface v_if;
    `Func_new("FIFO_config")
endclass //FIFO_config
endpackage
```

/// FIFO_reset_sequence.sv

```
package rst_sequence_pkg;
import shared_pkg::*;
import sequenceItem_pkg::*;
import uvm_pkg::*;
`include "defines/defines.svh"
`include "uvm_macros.svh"
class FIFO_reset_sequence extends uvm_sequence #(FIFO_sequenceItem);
    `uvm_object_utils(FIFO_reset_sequence)
    `Func_new("FIFO_reset_sequence")
    FIFO_sequenceItem item;
    // Main task
    task body;
        // Creat seq_item
        item = `create_obj(FIFO_sequenceItem, "item")
        `OFF_ALL
        repeat(5) begin
            start_item(item);
            item.data_in = 0; item.wr_en = 0; item.rd_en = 0;
            item.rst_n = 0;
            finish_item(item);
        end
    endtask
endclass //FIFO_reset_sequence extends uvm_sequence #(FIFO_sequenceItem)
endpackage
```

/// FIFO_write_only_sequence.sv

```
package write_only_sequence_pkg;
import shared_pkg::*;
import uvm_pkg::*;
import sequenceItem_pkg::*;
`include "uvm_macros.svh"
`include "defines/defines.svh"

class FIFO_write_only_sequence extends uvm_sequence #(FIFO_sequenceItem);
    `uvm_object_utils(FIFO_write_only_sequence)
    `Func_new("FIFO_write_only_sequence")
    FIFO_sequenceItem item;
    // Main task
    task body();
        if (isWriteTOfull)
            repeat(DEPTH_LOOP) begin
                item = `create_obj(FIFO_sequenceItem, "item")  // Creat seq_item
                ////////////////////////////
                //  edit constraint mode  //
                //    CONSTRAINT RULES    //
                ////////////////////////////
                `OFF_ALL
                `ON(CON_RESET)
                `ON(CON_W)
                start_item(item);
                assert (item.randomize());
                finish_item(item);
            end
        else
            repeat(BEFORE_ALMOST) begin
                item = `create_obj(FIFO_sequenceItem, "item")  // Creat seq_item
                `OFF_ALL
                `ON(CON_RESET)
                `ON(CON_W)
                start_item(item);
                assert (item.randomize());
                finish_item(item);
            end
        isWriteTOfull = 0;
    endtask
endclass //FIFO_write_only_sequence extends uvm_sequence #(FIFO_sequenceItem)
endpackage
```

/// FIFO_read_only_sequence.sv

```
package read_only_sequence_pkg;
import shared_pkg::*;
import uvm_pkg::*;
import sequenceItem_pkg::*;
`include "uvm_macros.svh"
`include "defines/defines.svh"

class FIFO_read_only_sequence extends uvm_sequence #(FIFO_sequenceItem);
    `uvm_object_utils(FIFO_read_only_sequence)

    `Func_new("FIFO_read_only_sequence")
    FIFO_sequenceItem item;
    // Main task
    task body();
        if (isFirstRead)
            repeat(BEFORE_ALMOST) begin
                item = `create_obj(FIFO_sequenceItem, "item")  // Creat seq_item
                `OFF_ALL
                `ON(CON_RESET)
                `ON(CON_R)

                start_item(item);
                assert (item.randomize());
                finish_item(item);
            end
        else
            repeat(DEPTH_LOOP) begin
                item = `create_obj(FIFO_sequenceItem, "item")  // Creat seq_item
                `OFF_ALL
                `ON(CON_RESET)
                `ON(CON_R)

                start_item(item);
                assert (item.randomize());
                finish_item(item);
            end

        isFirstRead = 0;
    endtask
endclass //FIFO_read_only_sequence extends uvm_sequence #(FIFO_sequenceItem)
endpackage
```

/// FIFO_write_read_sequence.sv

```
package write_read_sequence_pkg;
import shared_pkg::*;
import uvm_pkg::*;
import sequenceItem_pkg::*;
`include "uvm_macros.svh"
`include "defines/defines.svh"

class FIFO_write_read_sequence extends uvm_sequence #(FIFO_sequenceItem);
    `uvm_object_utils(FIFO_write_read_sequence)

    `Func_new("FIFO_write_read_sequence")

    FIFO_sequenceItem item;
    // Main task
    task body();
        if (isFIRST_DIST_FINITH) begin
            WR_EN_ON_DIST = 30;
            RD_EN_ON_DIST = 70;
        end
        repeat(HOBBIT_LOOP) begin
            item = `create_obj(FIFO_sequenceItem,"item")  // Creat seq_item
            `OFF_ALL
            `ON(CON_RESET)
            `ON(CON_W_R)
            `ON(CON_DATA_ONE_BIT)
            start_item(item);
            assert (item.randomize());
            finish_item(item);
        end
        repeat(BOOM_LOOP) begin
            item = `create_obj(FIFO_sequenceItem,"item")  // Creat seq_item
            `OFF_ALL
            `ON(CON_RESET)
            `ON(CON_W_R)
            start_item(item);
            assert (item.randomize());
            finish_item(item);
        end
    endtask
endclass //FIFO_write_read_sequence extends uvm_sequence #(FIFO_sequenceItem)
endpackage
```

/// FIFO_sync_toggle_sequence.sv

```
package sync_toggle_sequence_pkg;
import shared_pkg::*;
import uvm_pkg::*;
import sequenceItem_pkg::*;
`include "uvm_macros.svh"
`include "defines/defines.svh"

class FIFO_sync_toggle_sequence extends uvm_sequence #(FIFO_sequenceItem);
    `uvm_object_utils(FIFO_sync_toggle_sequence)
    `Func_new("FIFO_sync_toggle_sequence")
    FIFO_sequenceItem item;
    // Main task
    task body();
        for (int i = WR_START_VALUE_OF_TOGGLE; i<GIANT_LOOP; i++) begin
            item = `create_obj(FIFO_sequenceItem, "item")  // Creat seq_item
            `OFF_ALL
            `ON(CON_RESET)
            `ON(CON_TOGGLE)

            start_item(item);
            assert (item.randomize());
            item.wr_en = i%2;
            item.rd_en = i%2;
            finish_item(item);
        end
    endtask
endclass //FIFO_sync_toggle_sequence extends uvm_sequence #(FIFO_sequenceItem)
endpackage
```

/// FIFO_Async_toggle_sequence.sv

```
package Async_toggle_sequence_pkg;
import shared_pkg::*;
import uvm_pkg::*;
import sequenceItem_pkg::*;
`include "uvm_macros.svh"
`include "defines/defines.svh"

class FIFO_Async_toggle_sequence extends uvm_sequence #(FIFO_sequenceItem);
    `uvm_object_utils(FIFO_Async_toggle_sequence)

    `Func_new("FIFO_Async_toggle_sequence")

    FIFO_sequenceItem item;
    // Main task
    task body();
        for (int i = WR_START_VALUE_OF_TOGGLE; i<GIANT_LOOP; i++) begin
            item = `create_obj(FIFO_sequenceItem, "item")  // Creat seq_item
            `OFF_ALL
            `ON(CON_RESET)
            `ON(CON_DATA_ONE_BIT)

            start_item(item);
            assert (item.randomize());
            item.wr_en = i%2;
            item.rd_en = !(i%2);
            finish_item(item);
        end
    endtask
endclass //FIFO_Async_toggle_sequence extends uvm_sequence #(FIFO_sequenceItem)
endpackage
```

/// FIFO_random_sequence.sv

```
package random_sequence_pkg;
import shared_pkg::*;
import uvm_pkg::*;
import sequenceItem_pkg::*;
`include "uvm_macros.svh"
`include "defines/defines.svh"
class FIFO_random_sequence extends uvm_sequence #(FIFO_sequenceItem);
    `uvm_object_utils(FIFO_random_sequence)
    `Func_new("FIFO_random_sequence")
    FIFO_sequenceItem item;
    // Main task
    task body();
        repeat(BOOM_LOOP) begin
            item = `create_obj(FIFO_sequenceItem, "item")  // Creat seq_item
            `OFF_ALL
            `ON(CON_RESET)
            `ON(CON_DATA_MAX_ZERO)
            start_item(item);
            assert (item.randomize());
            finish_item(item);
        end
    endtask
endclass //FIFO_random_sequence extends uvm_sequence #(FIFO_sequenceItem)
endpackage
```

/// driver.sv

```
`timescale 1ps/1ps
package driver_pkg;
import shared_pkg::*;
import sequenceItem_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
`include "defines/defines.svh"
`define create_obj(type, name) type::type_id::create(name);

// driver class
class FIFO_driver extends uvm_driver #(FIFO_sequenceItem);
    `uvm_component_utils(FIFO_driver)
    virtual FIFO_interface v_if;
    FIFO_sequenceItem stim_seq_item;

    `Func_new_p("FIFO_driver")

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever begin
            stim_seq_item = `create_comp(FIFO_sequenceItem, "stim_seq_item")
            seq_item_port.get_next_item(stim_seq_item);
        // assigned inputs to interface
            v_if.rst_n   = stim_seq_item.rst_n;
            v_if.wr_en   = stim_seq_item.wr_en;
            v_if.rd_en   = stim_seq_item.rd_en;
            v_if.data_in = stim_seq_item.data_in;
            @(negedge v_if.clk);
        // valus driven
            seq_item_port.item_done();
            `uvm_info("run_phase_driver", stim_seq_item.print_DUT(), UVM_FULL)
        end
    endtask //run_phase
endclass //FIFO_driver extends uvm_driver

endpackage
```

/// monitor.sv

```
package monitor_pkg;
import shared_pkg::*;
import sequenceItem_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
`include "defines/defines.svh"
class FIFO_monitor extends uvm_monitor;
    `uvm_component_utils(FIFO_monitor)
    virtual FIFO_interface v_if;
    FIFO_sequenceItem mon_seq_item;
    uvm_analysis_port #(FIFO_sequenceItem) mon_port; // monitor is a port
    `Func_new_p("FIFO_monitor")
    function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        mon_port = new("mon_port", this);
    endfunction

    task run_phase(uvm_phase phase);
      super.run_phase(phase);
      forever begin
        mon_seq_item = `create_comp(FIFO_sequenceItem, "mon_seq_item")
        @(negedge v_if.clk);
        // assigned inputs to interface
            mon_seq_item.rst_n   = v_if.rst_n;
            mon_seq_item.wr_en   = v_if.wr_en;
            mon_seq_item.rd_en   = v_if.rd_en;
            mon_seq_item.data_in = v_if.data_in;
        // assigned outputs to driver
            mon_seq_item.data_out    = v_if.data_out;
            mon_seq_item.wr_ack      = v_if.wr_ack;
            mon_seq_item.overflow    = v_if.overflow;
            mon_seq_item.full        = v_if.full;
            mon_seq_item.empty       = v_if.empty;
            mon_seq_item.almostfull  = v_if.almostfull;
            mon_seq_item.almostempty = v_if.almostempty;
            mon_seq_item.underflow   = v_if.underflow;
        // assigned refrence to driver
            mon_seq_item.data_out_ref    = v_if.data_out_ref;
            mon_seq_item.wr_ack_ref      = v_if.wr_ack_ref;
            mon_seq_item.overflow_ref    = v_if.overflow_ref;
            mon_seq_item.full_ref        = v_if.full_ref;
            mon_seq_item.empty_ref       = v_if.empty_ref;
            mon_seq_item.almostfull_ref  = v_if.almostfull_ref;
            mon_seq_item.almostempty_ref = v_if.almostempty_ref;
            mon_seq_item.underflow_ref   = v_if.underflow_ref;
        mon_port.write(mon_seq_item); // that's mean that monitor will send the data
        `uvm_info("run_phase_monitor", mon_seq_item.print_DUT(), UVM_FULL)
      end
    endtask //run_pha
endclass //FIFO_monitor extends uvm_monitor
endpackage
```

/// sequencer.sv

```
package sequencer_pkg;
import uvm_pkg::*;
import sequenceItem_pkg::*;
`include "uvm_macros.svh"
`include "defines/defines.svh"

// sequencer class
class sequencer extends uvm_sequencer #(FIFO_sequenceItem);
    `uvm_component_utils(sequencer)
    `Func_new_p("sequencer")
endclass //sequencer extends uvm_sequencer #(FIFO_sequenceItem)
endpackage
```

/// agent.sv

```
`timescale 1ps/1ps
package agent_pkg;
import shared_pkg::*;
import config_pkg::*;
import driver_pkg::*;
import sequencer_pkg::*;
import sequenceItem_pkg::*;
import monitor_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
`include "defines/defines.svh"
//`define create_obj(type, name) type::type_id::create(name, this);

// agent class
class FIFO_agent extends uvm_agent;
    `uvm_component_utils(FIFO_agent)
    sequencer sqr; // mange data transfer
    FIFO_driver drv; // inside agent
    FIFO_monitor mon; // inside agent
    FIFO_config cfg; // get the data of interface
    uvm_analysis_port #(FIFO_sequenceItem) agt_port; // agent is a port

    `Func_new_p("FIFO_agent")

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

        if (!uvm_config_db#(FIFO_config)::get(this, "", "CFG", cfg))
            `uvm_fatal("build_phase", "DRIVER - Unable to get config");

        sqr = `create_comp(sequencer, "sqr")
        drv = `create_comp(FIFO_driver, "drv")
        mon = `create_comp(FIFO_monitor, "mon")
        agt_port = new("agt_port", this);
    endfunction

    function void connect_phase(uvm_phase phase);
        drv.v_if = cfg.v_if;
        mon.v_if = cfg.v_if;
        drv.seq_item_port.connect(sqr.seq_item_export);
        mon.mon_port.connect(agt_port);
    endfunction //connect_phase
endclass //FIFO_agent extends uvm_agent
endpackage
```

/// coverage_collector.sv

```
package coverage_collector_pkg;
import agent_pkg::*;
import shared_pkg::*;
import sequencer_pkg::*;
import sequenceItem_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
`include "defines/defines.svh"

class FIFO_coverage extends uvm_component;
    `uvm_component_utils(FIFO_coverage)

    uvm_analysis_export #(FIFO_sequenceItem) cov_export; // coverage export
    uvm_tlm_analysis_fifo #(FIFO_sequenceItem) cov_fifo; // coverage fifo
    FIFO_sequenceItem cov_seq_item;
```

```
////////////////////////////////
//      begin Coverage Group    //
////////////////////////////////

covergroup CVG;
    // rst_n coverage
        rst_cp: coverpoint cov_seq_item.rst_n{
                bins active = {0};
                bins inactive = {1};
                bins inactive_to_active = (1 => 0);
                bins active_to_inactive = (0 => 1);
        }

    // write and read enable signal coverpoint
        wr_en_cp:        coverpoint cov_seq_item.wr_en{
            bins active = {1};
            bins inactive = {0};
        }
        rd_en_cp:        coverpoint cov_seq_item.rd_en{
            bins active = {1};
            bins inactive = {0};
        }

    // data_out bus coverpoint
        data_out_cp:     coverpoint cov_seq_item.data_out{
            bins one_bit_H[] = one_bit_high;
            bins zero = {ZERO};
            bins max = {MAX};
            bins others = default;
        }

    // outputs signals coverpoint
        wr_ack_cp:       coverpoint cov_seq_item.wr_ack{
            bins active = {1};
            bins inactive = {0};
            bins inactive_to_active = (0 => 1);
            bins active_to_inactive = (1 => 0);
        }
        full_cp:         coverpoint cov_seq_item.full{
            bins active = {1};
            bins inactive = {0};
            bins active_to_inactive = (1 => 0);
            bins inactive_to_active = (0 => 1);
        }
        empty_cp:        coverpoint cov_seq_item.empty{
            bins active = {1};
            bins inactive = {0};
            bins active_to_inactive = (1 => 0);
            bins inactive_to_active = (0 => 1);
        }
        almostfull_cp:   coverpoint cov_seq_item.almostfull{
            bins active = {1};
            bins inactive = {0};
            bins active_to_inactive = (1 => 0);
            bins inactive_to_active = (0 => 1);
        }
        almostempty_cp:  coverpoint cov_seq_item.almostempty{
            bins active = {1};
            bins inactive = {0};
            bins active_to_inactive = (1 => 0);
            bins inactive_to_active = (0 => 1);
        }
        underflow_cp:    coverpoint cov_seq_item.underflow{
            bins active = {1};
            bins inactive = {0};
        }
        overflow_cp:     coverpoint cov_seq_item.overflow{
            bins active = {1};
            bins inactive = {0};
        }

    // Cross coverage
    // A -> refear to Active
    // I -> refear to Inactive

    // wr_ack signal
        // reset
        ack_rst_cross: cross rst_cp, wr_ack_cp {
            bins rst_active_ack_inactive = binsof(rst_cp.active) && binsof(wr_ack_cp.inactive);
            option.cross_auto_bin_max = 0;
        }

        // wr_en and rd_en ** requirement **
        ack_wr_rd_cross: cross wr_ack_cp, wr_en_cp, rd_en_cp{
            bins activate_ack_wr_inactive = binsof(wr_en_cp.inactive) && binsof(wr_ack_cp.active_to_inactive);
            bins deactivate_ack_wr_active = binsof(wr_en_cp.active) && binsof(wr_ack_cp.inactive_to_active);
            bins ack_inactive_wr_inactive = binsof(wr_en_cp.inactive) && binsof(wr_ack_cp.inactive);

            bins deactivate_ack_rd_active = binsof(rd_en_cp.active) && binsof(wr_ack_cp.active_to_inactive);
            bins deactivate_ack_wr_active_rd_active = binsof(rd_en_cp.active)&&  binsof(wr_en_cp.active) && binsof(wr_ack_cp.active_to_inactive);
            option.cross_auto_bin_max = 0;
        }

        // full and wr_en
        // crossing wr_ack with full when wr_ack is active and full is active
        // crossing wr_ack with full when full rose and wr_ack fell
```

```
    ack_full_wr_cross: cross wr_ack_cp, wr_en_cp, full_cp{
        bins ack_active_wr_active_full_inactive = binsof(wr_ack_cp.active)
                                && binsof(wr_en_cp.active)
                                && binsof(full_cp.inactive);
        bins ack_active_full_inactive = binsof(wr_ack_cp.active) && binsof(full_cp.inactive);
        bins activated_full_activated_ack = binsof(wr_ack_cp.inactive_to_active) && binsof(full_cp.inactive_to_active);
        option.cross_auto_bin_max = 0;
    }

    // empty
    ack_empty_cross: cross empty_cp, wr_ack_cp {
        bins deactivated_empty_wr_active = binsof(wr_ack_cp.active) && binsof(empty_cp.active_to_inactive);
        option.cross_auto_bin_max = 0;
    }

    // almostempty
    ack_almostempty_cross: cross almostempty_cp, wr_ack_cp {
        bins ack_active_almostempty_inactive = binsof(wr_ack_cp.active) && binsof(almostempty_cp.inactive);
        option.cross_auto_bin_max = 0;
    }

// full signal
    // rst transaction
    rst_full_cross: cross full_cp, rst_cp{
        bins deactivate_full_activate_rst = binsof(full_cp.active_to_inactive) && binsof(rst_cp.inactive_to_active);
        option.cross_auto_bin_max = 0;
    }
    // wr_en and rd_en ** requirement **
    full_cross: cross full_cp, wr_en_cp, rd_en_cp{
        bins activate_full_wr_active = binsof(full_cp.inactive_to_active) && binsof(wr_en_cp.active);
        bins full_active_wr_active = binsof(full_cp.active) && binsof(wr_en_cp.active);
        bins deactivate_full_rd_active = binsof(full_cp.active_to_inactive) && binsof(rd_en_cp.active);
        option.cross_auto_bin_max = 0;
    }

    // almostfull transaction
    // crossing to detect when almostfull trans from active to inactive and full from inactive to active
    // and oppesite operation
    almostfull_full_cross: cross almostfull_cp, full_cp{
        bins trans_almostfull_to_full = binsof(almostfull_cp.active_to_inactive) && binsof(full_cp.inactive_to_active);
        bins trans_full_to_almostfull = binsof(almostfull_cp.inactive_to_active) && binsof(full_cp.active_to_inactive);
        option.cross_auto_bin_max = 0;
    }

    // overflow
    // crossing to detect when overflow and full both active
    full_overflow_cross:   cross overflow_cp, full_cp{
        bins overflow_full_both_active = binsof(overflow_cp.active) && binsof(full_cp.active);
        option.cross_auto_bin_max = 0;
    }

// empty signal
    // rst
    rst_empty_cross: cross empty_cp, rst_cp {
        option.cross_auto_bin_max = 0;
        bins rst_empty = binsof(empty_cp.inactive) && binsof(rst_cp.active);
        bins deactivate_rst_activate_empty = binsof(rst_cp.active_to_inactive) && binsof(empty_cp.inactive_to_active);
    }

    // almostempty trans
    // crossing to detect when almostempty trans from active to inactive and empty from inactive to active
    // and oppesite operation
    almostempty_empty_cross: cross almostempty_cp, empty_cp{
        bins trans_almostempty_to_empty = binsof(almostempty_cp.active_to_inactive) && binsof(empty_cp.inactive_to_active);
        bins trans_empty_to_almostempty = binsof(almostempty_cp.inactive_to_active) && binsof(empty_cp.active_to_inactive);
        option.cross_auto_bin_max = 0;
    }

    // rd_en and wr_en
    empty_cross: cross empty_cp, wr_en_cp, rd_en_cp{
        bins activate_empty_rd_active = binsof(empty_cp.inactive_to_active) && binsof(rd_en_cp.active);
        bins empty_active_rd_active = binsof(empty_cp.active) && binsof(rd_en_cp.active);
        bins deactivate_empty_wr_active = binsof(empty_cp.active_to_inactive) && binsof(wr_en_cp.active);
        option.cross_auto_bin_max = 0;
    }

    // underflow
    // crossing to detect when underflow and empty both active
    empty_underflow_cross: cross underflow_cp, empty_cp{
        bins underflow_empty = binsof(underflow_cp.active) && binsof(empty_cp.active);
        option.cross_auto_bin_max = 0;
    }

// overflow signal
    // rst
    rst_overflow_cross: cross rst_cp, overflow_cp {
        bins rst_active_ack_inactive = binsof(rst_cp.active) && binsof(overflow_cp.inactive);
        option.cross_auto_bin_max = 0;
    }

    // wr_en
    // crossing to detect when overflow and write enabe both active
    wr_overflow_cross: cross wr_en_cp, overflow_cp{
        bins both_high = binsof(wr_en_cp.active) && binsof(overflow_cp.active);
        bins overflow_high = binsof(wr_en_cp.active) && binsof(overflow_cp.inactive);
        option.cross_auto_bin_max = 0;
```

```
        }

    // underflow signal
        // rst
        rst_underflow_cross: cross rst_cp, underflow_cp {
            bins rst_ack = binsof(rst_cp.active) && binsof(underflow_cp.inactive);
            option.cross_auto_bin_max = 0;
        }

        // rd_en
        // crossing to detect when underflow and read enabe both active
        rd_underflow_cross: cross rd_en_cp, underflow_cp{
            bins both_high = binsof(rd_en_cp.active) && binsof(underflow_cp.active);
            bins rd_high = binsof(rd_en_cp.active) && binsof(underflow_cp.inactive);
            option.cross_auto_bin_max = 0;
        }

    // almostempty signal
        // rst
        rst_almostempty_cross: cross rst_cp, almostempty_cp{
            bins rst_almostempty = binsof(rst_cp.active) && binsof(almostempty_cp.inactive);
            option.cross_auto_bin_max = 0;
        }

        // rd_en and wr_en
        almostempty_cross:  cross wr_en_cp, rd_en_cp, almostempty_cp{
            bins write_almost_active = binsof(wr_en_cp.active) && binsof(almostempty_cp.active);
            bins write_Active_almost_inactive = binsof(wr_en_cp.active) && binsof(almostempty_cp.inactive);
            bins almost_read_active = binsof(rd_en_cp.active) && binsof(almostempty_cp.active);
            bins read_active_almost_inactive = binsof(rd_en_cp.active) && binsof(almostempty_cp.inactive);
            bins activate_almost_write_active = binsof(wr_en_cp.active) && binsof(almostempty_cp.inactive_to_active);
            bins activate_almost_read_active = binsof(rd_en_cp.active) && binsof(almostempty_cp.inactive_to_active);
            option.cross_auto_bin_max = 0;
        }

    // almostfull signal
        // rst
        rst_almostfull_cross: cross rst_cp, almostfull_cp{
            bins rst_almostfull = binsof(rst_cp.active) && binsof(almostfull_cp.inactive);
            option.cross_auto_bin_max = 0;
        }

        // wr_en and rd_en
        almostfull_cross:   cross wr_en_cp, rd_en_cp, almostfull_cp{
            bins wr_active_almost_active = binsof(wr_en_cp.active) && binsof(almostfull_cp.active);
            bins wr_active_almost_inactive = binsof(wr_en_cp.active) && binsof(almostfull_cp.inactive);
            bins rd_active_almost_active = binsof(rd_en_cp.active) && binsof(almostfull_cp.active);
            bins rd_active_almost_inactive = binsof(rd_en_cp.active) && binsof(almostfull_cp.inactive);
            option.cross_auto_bin_max = 0;
        }

    // data_out bus
        // reset
        rst_data_out_cross: cross rst_cp, data_out_cp {
            bins rst_data = binsof(rst_cp.active) && binsof(data_out_cp.zero);
            option.cross_auto_bin_max = 0;
        }

        // rd_en and wr_en ** requirement **
        data_out_cross: cross data_out_cp, wr_en_cp, rd_en_cp;

    // Crossing onley read and write
    rd_wr_cross: cross rd_en_cp, wr_en_cp;
endgroup

////////////////////////////////
//      finish Coverage Group    //
////////////////////////////////

// Methods
    function new(string name = "FIFO_coverage", uvm_component parent = null);
        super.new(name, parent);
        CVG = new();
    endfunction //new()

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

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

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever begin
            cov_fifo.get(cov_seq_item);
            CVG.sample();
        end
    endtask
endclass //FIFO_coverage extends uvm_component
endpackage
```

/// scoreboard.sv

```
package scoreboard_pkg;
import agent_pkg::*;
import shared_pkg::*;
import sequenceItem_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
`include "defines/defines.svh"

class FIFO_scoreboard extends uvm_scoreboard;
    `uvm_component_utils(FIFO_scoreboard)
    uvm_analysis_export #(FIFO_sequenceItem) sb_export; // scoreboard export
    uvm_tlm_analysis_fifo #(FIFO_sequenceItem) sb_fifo; // scoreboard fifo
    FIFO_sequenceItem sb_seq_item;

    // error and correct counter
    int correct_counter = 0;
    int error_counter = 0;

    `Func_new_p("FIFO_scoreboard")

    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(sb_seq_item);
            //// Checking //////
            Checking_task(sb_seq_item);
        end
    endtask

    task Checking_task(FIFO_sequenceItem chk_item);
        if (
            chk_item.data_out!=chk_item.data_out_ref
            || chk_item.wr_ack!=chk_item.wr_ack_ref
            || chk_item.full!=chk_item.full_ref
            || chk_item.empty!=chk_item.empty_ref
            || chk_item.underflow!=chk_item.underflow_ref
            || chk_item.almostempty!=chk_item.almostempty_ref
            || chk_item.almostfull!=chk_item.almostfull_ref
            || chk_item.overflow!=chk_item.overflow_ref
        ) begin
            error_counter++;
            `uvm_error("scoreboard",$sformatf("%0s\n%0s,",chk_item.print_DUT(), chk_item.print_REF()))
        end else begin
            correct_counter++;
        end
    endtask //Checking_task

    function void report_phase(uvm_phase phase);
        super.report_phase(phase);
        `uvm_info("report_phase", $sformatf("Total correct transaction: %0d", correct_counter), UVM_LOW)
        `uvm_info("report_phase", $sformatf("Total faild transaction: %0d", error_counter), UVM_LOW)
    endfunction
endclass //FIFO_scoreboard extends uvm_scoreboard
endpackage
```

/// env.sv

```
`timescale 1ps/1ps
package env_pkg;
import shared_pkg::*;
import scoreboard_pkg::*;
import coverage_collector_pkg::*;
import agent_pkg::*;
import uvm_pkg::*;
`include "defines/defines.svh"
`include "uvm_macros.svh"
//`define create_obj(type, name) type::type_id::create(name, this);

// Environment class
class FIFO_env extends uvm_env;
    `uvm_component_utils(FIFO_env)

    FIFO_scoreboard sb;
    FIFO_coverage cov;
    FIFO_agent agt;
```

```
    // declare new() function of parent uvm_env
    `Func_new_p("FIFO_env")

    // build phase function and send the parameter phase to parent uvm_env
    function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        // create scoreboard, coverage and agent
        // if we have more than one I should change name (first parameter)
        agt = `create_comp(FIFO_agent, "agt")
        sb = `create_comp(FIFO_scoreboard, "sb")
        cov = `create_comp(FIFO_coverage, "cov")
    endfunction

    function void connect_phase(uvm_phase phase);
        agt.agt_port.connect(sb.sb_export);
        agt.agt_port.connect(cov.cov_export);
    endfunction
endclass //FIFO_env extends uvm_env
endpackage
```

/// test.sv

```
package test_pkg;
import shared_pkg::*;
import env_pkg::*;
import config_pkg::*;

import write_only_sequence_pkg::*;
import read_only_sequence_pkg::*;
import write_read_sequence_pkg::*;
import sync_toggle_sequence_pkg::*;
import Async_toggle_sequence_pkg::*;
import rst_sequence_pkg::*;
import random_sequence_pkg::*;

import sequenceItem_pkg::*;
import uvm_pkg::*;
`include "defines/defines.svh"
`include "uvm_macros.svh"

class FIFO_test extends uvm_test;
    `uvm_component_utils(FIFO_test)
    FIFO_env env;
    FIFO_config cfg;

    FIFO_reset_sequence reset_seq;
    FIFO_write_only_sequence write_seq;
    FIFO_read_only_sequence  read_seq;
    FIFO_write_read_sequence both_seq;
    FIFO_sync_toggle_sequence sync_toggle_seq;
    FIFO_Async_toggle_sequence async_toggle_seq;
    FIFO_random_sequence random_seq;

    // declare new() function of parent uvm_test
    `Func_new_p("FIFO_test")

    // build phase function and send the parameter phase to parent uvm_test
        function void build_phase(uvm_phase phase);
            uvm_factory factory = uvm_factory::get();
            super.build_phase(phase);
            //set_type_override_by_type(FIFO_sequenceItem::get_type(), FIFO_sequenceItem_valid::get_type());
            factory.print();
            env  = `create_comp(FIFO_env, "env")
            cfg  = `create_comp(FIFO_config, "cfg")
            // sequences
            reset_seq     = `create_comp(FIFO_reset_sequence, "reset_seq")
            write_seq     = `create_comp(FIFO_write_only_sequence, "write_seq")
            read_seq      = `create_comp(FIFO_read_only_sequence, "read_seq")
            both_seq      = `create_comp(FIFO_write_read_sequence, "both_seq")
            sync_toggle_seq  = `create_comp(FIFO_sync_toggle_sequence, "sync_toggle_seq")
            async_toggle_seq = `create_comp(FIFO_Async_toggle_sequence, "async_toggle_seq")
            random_seq    = `create_comp(FIFO_random_sequence, "random_seq")

             if (!uvm_config_db#(virtual FIFO_interface)::get(this, "", "INTERFACE", cfg.v_if))
                    `uvm_fatal("build_phase", "TEST - Unable to get config");

            uvm_config_db#(FIFO_config)::set(this, "*", "CFG", cfg);
        endfunction

    // run phase function to create UVM env
        task run_phase(uvm_phase phase);
            super.run_phase(phase);

            // raise and drop to start and finish of FIFO_test
            phase.raise_objection(this);
            #1; `uvm_info("run_phase", "Inside the slaby test DEBUG", UVM_DEBUG)

            ////////////////////////
            //    stimulus start    //
            ////////////////////////
            // rst seq
            `uvm_info("run_phase", "FIFO reset seq", UVM_LOW)
```

```
            reset_seq.start(env.agt.sqr);

            // LOCK RESET
            RESET_ACTIVE = 0;
            isWriteTOfull = 1;
            `uvm_info("run_phase", "FIFO write only seq", UVM_LOW)
            write_seq.start(env.agt.sqr);

            isFirstRead = 1;
            `uvm_info("run_phase", "FIFO read only seq", UVM_LOW)
            read_seq.start(env.agt.sqr);

            `uvm_info("run_phase", "FIFO write and read seq - WRITE DIST MORE THAN READ -", UVM_LOW)
            both_seq.start(env.agt.sqr);
            isFIRST_DIST_FINITH = 1;

            `uvm_info("run_phase", "FIFO sync toggle seq", UVM_LOW)
            sync_toggle_seq.start(env.agt.sqr);

            `uvm_info("run_phase", "FIFO read only seq", UVM_LOW)
            read_seq.start(env.agt.sqr);

            `uvm_info("run_phase", "FIFO write only seq", UVM_LOW)
            write_seq.start(env.agt.sqr);

            `uvm_info("run_phase", "FIFO async toggle seq", UVM_LOW)
            async_toggle_seq.start(env.agt.sqr);

            `uvm_info("run_phase", "FIFO read only seq", UVM_LOW)
            read_seq.start(env.agt.sqr);

            `uvm_info("run_phase", "FIFO write only seq", UVM_LOW)
            write_seq.start(env.agt.sqr);
            // UNLOCK RESET
            RESET_ACTIVE = 3;
            `uvm_info("run_phase", "FIFO write and read seq - READ DIST MORE THAN WRITE -", UVM_LOW)
            both_seq.start(env.agt.sqr);

            `uvm_info("run_phase", "FIFO reset seq", UVM_LOW)
            reset_seq.start(env.agt.sqr);

            `uvm_info("run_phase", "FIFO random seq", UVM_LOW)
            random_seq.start(env.agt.sqr);
            ////////////////////////
            //    stimulus finish    //
            ////////////////////////

            phase.drop_objection(this);
        endtask
endclass //FIFO_test extends uvm_test
endpackage
```

/// top.sv

```
`timescale 1ps/1ps
import uvm_pkg::*;
`include "uvm_macros.svh"
import test_pkg::*;
module top ();
    bit clk;

    initial begin
        forever #1 clk = ~clk;
    end

    FIFO_interface intf (clk);
    FIFO DUT (
        intf.data_in, intf.wr_en, intf.rd_en, clk, intf.rst_n,

        intf.full, intf.empty, intf.almostfull, intf.almostempty,
        intf.wr_ack, intf.overflow, intf.underflow, intf.data_out
    );

    FIFO_ref GLD (
        intf.data_in, intf.wr_en, intf.rd_en, clk, intf.rst_n,

        intf.full_ref, intf.empty_ref, intf.almostfull_ref, intf.almostempty_ref,
        intf.wr_ack_ref, intf.overflow_ref, intf.underflow_ref, intf.data_out_ref
    );

    bind FIFO FIFO_sva FIFO_sva_inst(
        intf.data_in, intf.wr_en, intf.rd_en, clk, intf.rst_n,

        intf.full, intf.empty, intf.almostfull, intf.almostempty,
        intf.wr_ack, intf.overflow, intf.underflow, intf.data_out
    );

    initial begin
        uvm_config_db#(virtual FIFO_interface)::set(null, "uvm_test_top", "INTERFACE", intf);
        run_test("FIFO_test");
    end
endmodule
```

/// ** Assertion definition ** ///

Macro	define
asrt_fn	assert final
cov_fn	Cover final
asrt_prp	cover property (@(posedge clk) disable iff(!rst_n) ());
cov_prp	assert property (@(posedge clk) disable iff(!rst_n) ());
same_seq	(rd_en && wr_en && !empty && !full)
A_A(Active1, Active2)	Active1 && Active2
A_I(Active, Inactive);	Active && ~Inactive

Internal signal Assertion

```
/////////////////////////////////////////////////////////////////////////////////////////////////
||                                                  ||  assert_count_rst:  `asrt_fn(count == 0); ||
|| whenever assert is activated this signal get low ||  assert_wr_ptr_rst: `asrt_fn(wr_ptr == 0);||
||                                                  ||  assert_rd_ptr_rst: `asrt_fn(rd_ptr == 0);||
/////////////////////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
|| When wr_en = 1 &rd_en=0 & full = 0, count increment by one  | count_inc: `asrt_prp(A_II(wr_en, rd_en, full) |=> ($past(count)+1 == count)); ||
|| when rd_en=1 & wr_en=0 & empty = 0, count decrement by one  | count_dec: `asrt_prp(A_II(rd_en, wr_en, empty)|=> ($past(count)-1 == count)); ||
|| when `same_seq excuted, count will not change               | count_noChange: `asrt_prp(`same_seq |=> ($past(count) == count));             ||
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
|| when wr_en = 1 & count less than FIFO_DEPTH, wr_ptr increment || inc_wr_ptr_assert: `asrt_prp(A_A(wr_en,(count < FIFO_DEPTH)) |=> inc_ptr(wr_ptr)); ||
|| when rd_en = 1 & count less than FIFO_DEPTH, rd_ptr increment || inc_rd_ptr_assert: `asrt_prp((rd_en && count != 0) |=> inc_ptr(rd_ptr) );          ||
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
|| whenever count is higher than or equal FIFO_DEPTH, full get high || full_count:`asrt_prp((count >= FIFO_DEPTH) |->  full);              ||
|| whenever count is equal FIFO_DEPTH-1, almostfull get high        || almostfull_count:`asrt_prp((count==FIFO_DEPTH-1) |-> almostfull); ||
|| whenever count is equal 1, almostempty get high                  || almostempty_count: `asrt_prp((count==1) |-> almostempty );        ||
|| whenever count is equal ZERO, empty get high                     || empty_count: `asrt_prp((count==ZERO) |-> empty );                 ||
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

Global Signal Assertion

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                    || assert_wr_akc_rst:    `asrt_fn(wr_ack == 0);       ||
//                                                    || assert_overflow_rst:  `asrt_fn(overflow == 0);     ||
//                                                    || assert_underflow_rst: `asrt_fn(underflow == 0);    ||
//                                                    || assert_data_out_rst:  `asrt_fn(data_out == 0);     ||
// whwnever reset is active, all this signal should be inactive || assert_full_rst:        `asrt_fn(full == 0);         ||
//                                                    || assert_almostfull_rst:  `asrt_fn(almostfull == 0); ||
//                                                    || assert_empty_rst:       `asrt_fn(empty == 0);      ||
//                                                    || assert_almostempty_rst: `asrt_fn(almostempty == 0);||
//////////////////////////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
|| when almostfull is high and wr_en = 1 & rd_en = 0, full get high    || full_from_almost: `asrt_prp(AA_I(almostfull,wr_en,rd_en) |=> full); ||
|| when full is high and rd_en =1, full will get low                   || full_inactive:    `asrt_prp(A_A(full,rd_en) |=> !full);               ||
|| when same_seq acheved, full won't change from previous clk.cycle    || full_noChange:    `asrt_prp( `same_seq |=> $past(full) == full);    ||
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////
|| wr_en=1&fifo not full, wr_ack = 1     || ack_active:   `asrt_prp(A_I(wr_en,full) |=> wr_ack);||
|| fifo is full, wr_ack=0, can't be high || ack_inactive: `asrt_prp( full |=> !wr_ack);         ||
//////////////////////////////////////////////////////////////////////////////
```

full=1 & rd_en=1, full fell and almostfull raise	almostfull_from_full: `asrt_prp(A_A(full,rd_en)|=> ($fell(full) && $rose(almostfull)))
when same_seq acheved, almostfull won't change	almostfull_noChange: `asrt_prp( `same_seq |=> $past(almostfull) == almostfull);
almostfull=1 & rd_en=0 and wr_en=0, almostfull Not active any more	almostfull_inactive: `asrt_prp(AA_I(almostfull,rd_en,wr_en) |=> !almostfull);

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
|| fifo is full and wr_en activated, overflow activated      || overflow_active:`asrt_prp(A_A(full,wr_en) |=> overflow);           ||
|| last.clk.cycle.overflow=1 & wr_en=0, overflow deactivated || overflow_wr_in: `asrt_prp(($past(overflow) && !wr_en) |=> !overflow);||
|| last.clk.cycle.overflow=1 & full=0, overflow deactivated  || overflow_Nfull: `asrt_prp(($past(overflow) && !full)  |=> !overflow);||
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

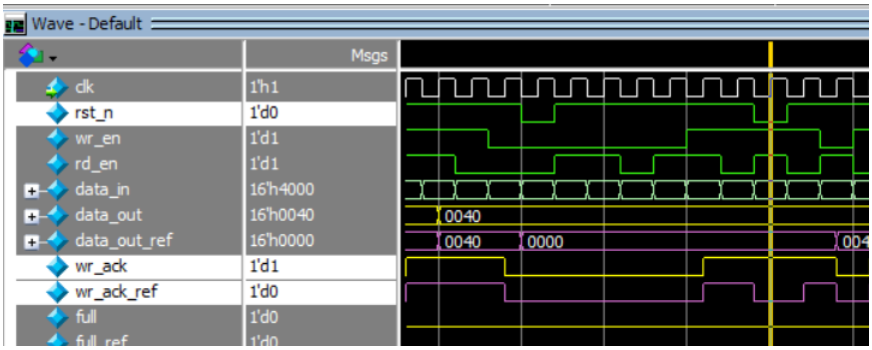
when same_seq acheved, almostfull won't change	almostempty_noChange: `asrt_prp( `same_seq |=> ($past(almostempty) == almostempty));
empty=1 & rd_en=1, empty fell and almostempty raise	almostempty_from_empty:`asrt_prp(A_A(empty,wr_en) |=> ($fell(empty) && $rose(almostempty))
almostempty=1 & wr_en=0 and rd_en=0, almostempty Not active any more	almostempty_inactive: `asrt_prp(AA_I(almostempty,wr_en,rd_en) |=> !almostempty)

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
|| when same_seq acheved, empty won't change            || empty_noChaneg:   `asrt_prp( `same_seq |=> ($past(empty) == empty) );||
|| almostempty=1 & rd_en=1 and wr_en=0, empty activated ||empty_from_almost: `asrt_prp(AA_I(almostempty,rd_en,wr_en) |=> empty);||
|| empty=1 & wr_en=1, empty deactivated                 ||empty_inactive:    `asrt_prp(A_A(empty,wr_en) |=> !empty);           ||
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
|| empty=1 & rd_en=1, underflow activated                  ||underflow_active:  `asrt_prp(A_A(empty,rd_en) |=> underflow);            ||
|| last.clk.cycle.underflow=1 & rd_en=0, overflow deactivated ||underflow_Nrd:     `asrt_prp(($past(underflow) && !rd_en)  |=> !underflow);||
|| last.clk.cycle.underflow=1 & empty=0, overflow deactivated ||underflow_Nempty:  `asrt_prp(($past(underflow) && !empty)  |=> !underflow);||
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

/// ** Bug_1: The signal wr_ack is sequential so it must be reset to zero when reset is activate ** ///

```
#    Time: 66 ns  Scope: scoreboard_pkg.FIFO_scoreboard.check_data File: package/scoreboard_pkg.sv Line: 38
# ** Error:                   66: Error - wr_ack_ref = 0, wr_ack = 1
```

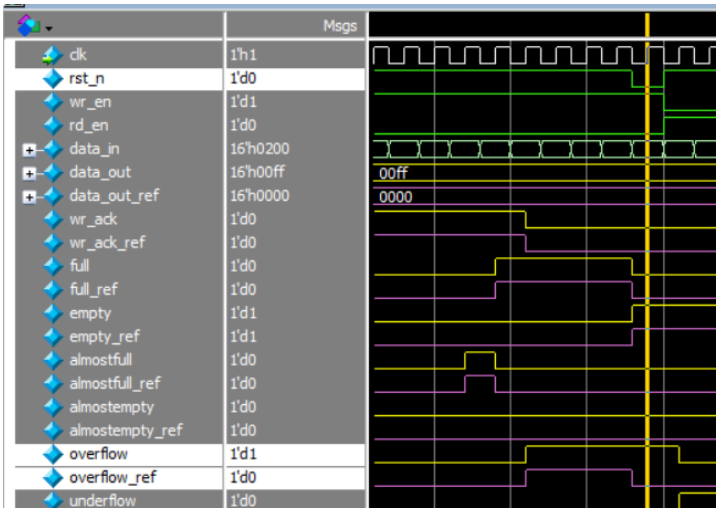


FIX:

```
    if (!F_if.rst_n) begin
        wr_ptr <= 0;
        F_if.wr_ack <= 0; //
    end
```

/// ** Bug_2: The signal overflow is sequential so it must be reset to zero when reset is activate ** ///

```
#    Time: 670 ns  Scope: scoreboard_pkg.FIFO_scoreboard.check_data File: package/scoreboard_pkg.sv Line: 38
# ** Error:                 670: Error - overflow_ref = 0, overflow = 1
```

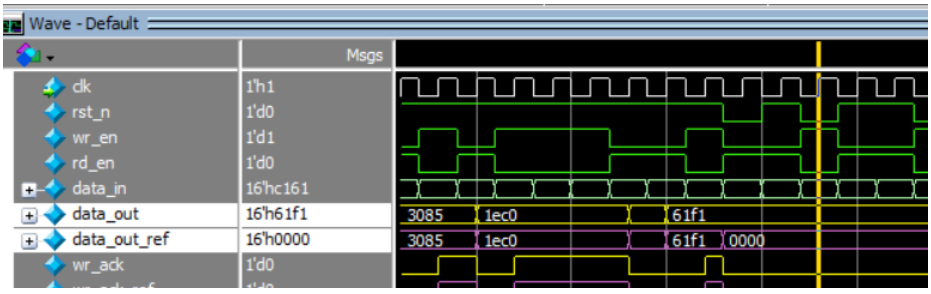


Fix:

```
if (!F_if.rst_n) begin
    wr_ptr <= 0;
    F_if.wr_ack <= 0; //
    F_if.overflow <= 0; //
end
```

/// ** Bug_3: The data_out bus is sequential so it must be reset to zero when reset is activate ** ///

```
# ** Error:                 388: Error - data_out_ref = 0, data_out = 4096
#    Time: 388 ns  Scope: scoreboard_pkg.FIFO_scoreboard.check_data File: package/scoreboard_pkg.sv Line: 38
```



FIX:

```
if (!F_if.rst_n) begin
    rd_ptr <= 0;
    F_if.underflow <= 0;// Fix
    F_if.data_out <= 0;// Fix
end
```

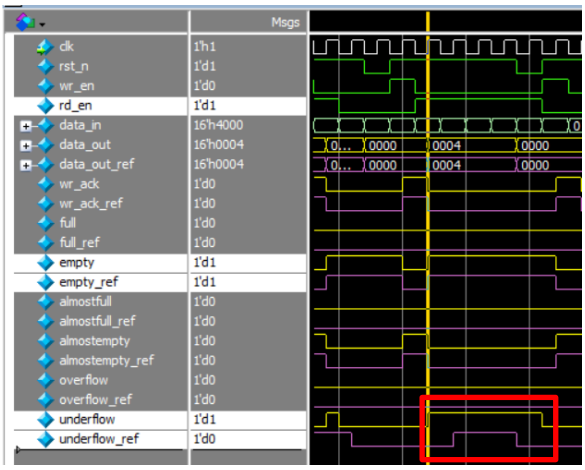
/// ** Bug_4: underflow must be sequential not combinational ** ///

Bug in code:

```
assign underflow = (empty && rd_en)? 1 : 0;
```

Bug in waveform:

```
#    Time: 530 ns  Scope: scoreboard_pkg.FIFO_scoreboard.check_data File: package/scoreboard_pkg.sv Line: 66
# ** Error:                  540: Error - underflow_ref = 0, underflow = 1
```



FIX:

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

```
// assign F_if.underflow = (F_if.empty && F_if.rd_en)? 1 : 0; //
```

/// ** Bug_4 cont.: The underflow is sequential so it must be reset to zero when reset is activate ** ///

```
always @(posedge F_if.clk or negedge F_if.rst_n) begin
    if (!F_if.rst_n) begin
        rd_ptr <= 0;
        F_if.underflow <= 0;// Fix
        F_if.data_out <= 0;// Fix
    end
```

/// ** Bug_5: Counter is not handle the case of wr_en and rd_en both are high ** ///

Bug in code:

```
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
```

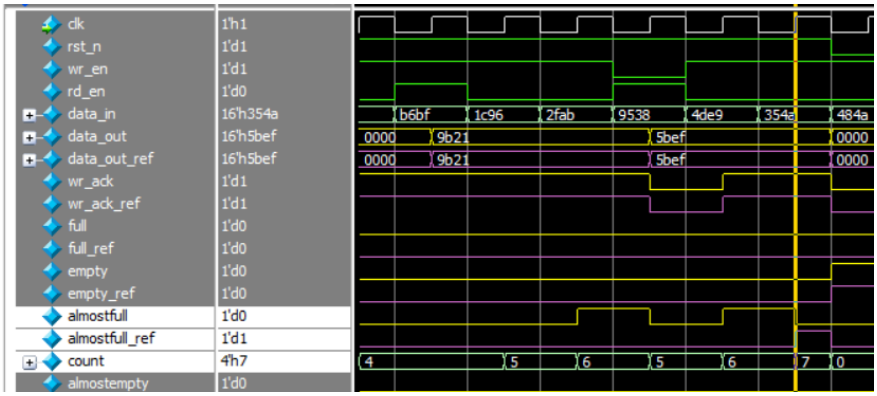
Fix:

```
        if (({F_if.wr_en, F_if.rd_en} == 2'b11) && F_if.full) // FIX
            count <= count - 1; // FIX
```

```
        else if (({F_if.wr_en, F_if.rd_en} == 2'b11) && F_if.empty) // FIX
            count <= count + 1; // FIX
        else if ( ({F_if.wr_en, F_if.rd_en} == 2'b10) && !F_if.full)
            count <= count + 1;
        else if ( ({F_if.wr_en, F_if.rd_en} == 2'b01) && !F_if.empty)
            count <= count - 1;
```

/// ** Bug_6: almostfull is high when internal signal “count” is less than FIFO_DEPTH by one** ///

```
# ** Error:                  158: Error - almostfull_ref = 0, almostfull = 1
#    Time: 158 ns  Scope: scoreboard_pkg.FIFO_scoreboard.check_data File: package/scoreboard_pkg.sv Line: 58
```




```
assign almostfull = (count == FIFO_DEPTH-2)? 1 : 0;
```

Fix:

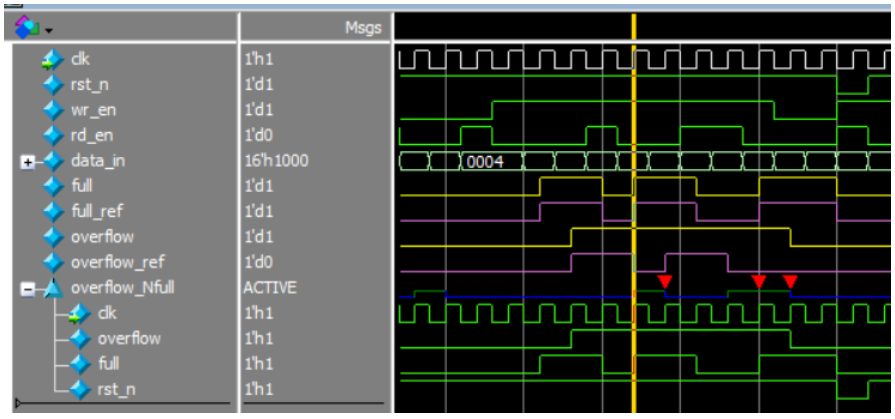
```
assign F_if.almostfull = (count == FIFO_DEPTH-1)? 1 : 0; //
```

/// ** Bug_7: overflow** ///

FIFO is full => inputs(wr_en = 1, rd_en = 1) then overflow gets high and FIFO no longer full

At next clk cycle inputs(wr_en = 1, rd_en = X) write operation will succeed but overflow still high

```
/top/dut/overflow_Nfull                Concurrent    SVA    on    172
```



Fix:

```
    else if (F_if.wr_en && count < FIFO_DEPTH) begin
        mem[wr_ptr] <= F_if.data_in;
        F_if.wr_ack <= 1;
        wr_ptr <= wr_ptr + 1;
        F_if.overflow <= 0; // FIX
    end
```

/// ** Bug_8: in if condition should be “&&” not “&” only to achieve 100% Condition Coverage ///**

```
----------------Focused Condition View-------------------
Line       31 Item    1  (F_if.full & F_if.wr_en)
Condition totals: 1 of 2 input terms covered = 50.00%

  Input Term   Covered  Reason for no coverage   Hint
 -----------  --------  -----------------------  --------------
   F_if.full         N  '_0' not hit             Hit '_0'
  F_if.wr_en         Y
```

```
        if (F_if.full & F_if.wr_en)
            F_if.overflow <= 1;
```

Fix:

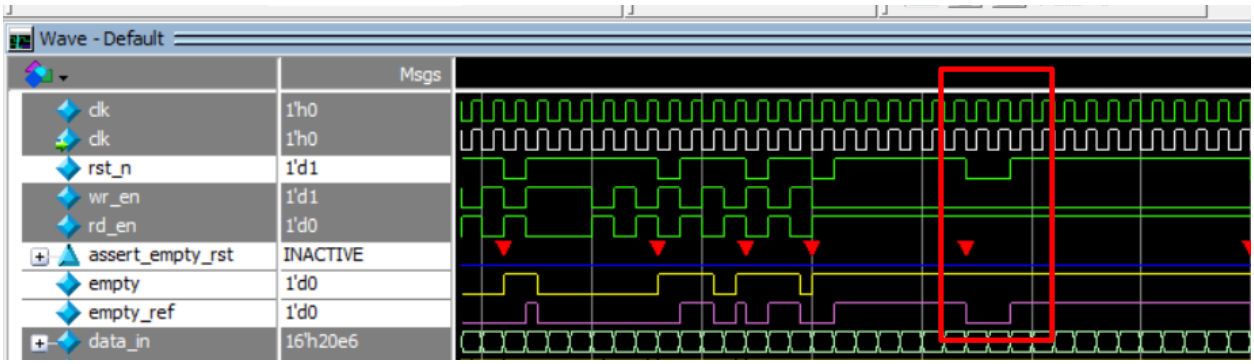
```
        if (F_if.full && F_if.wr_en)// FIX
            F_if.overflow <= 1;
```

```
----------------Focused Condition View-------------------
Line       31 Item    1  (F_if.full && F_if.wr_en)
Condition totals: 2 of 2 input terms covered = 100.00%

  Input Term   Covered  Reason for no coverage   Hint
 -----------  --------  -----------------------  --------------
   F_if.full         Y
  F_if.wr_en         Y
```

/// ** Bug_9: when rst_n is activated then empty signal must be zero ///**

```
** Error: Assertion error.
   Time: 764 ns  Scope: top.dut.rst_n_assert.assert_empty_rst File: design/FIFO.sv Line: 139
** Error: empty = 1, empty_ref = 0
   Time: 766 ns  Scope: scoreboard_pkg.FIFO_scoreboard.display_errors File: package/scoreboard_pkg.sv Line: 51
```



Fix:

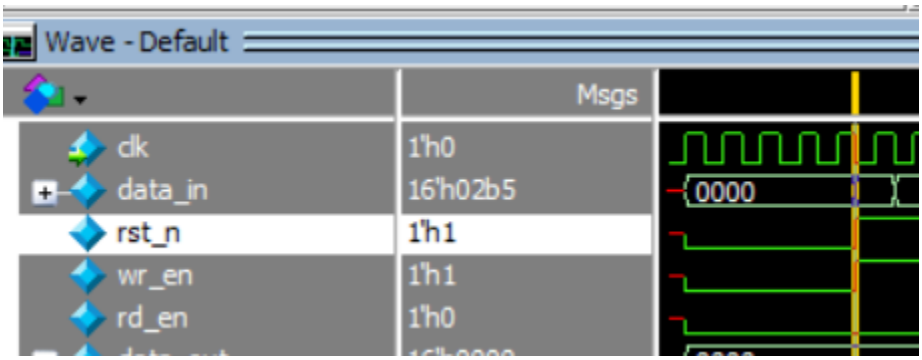
```
assign F_if.empty = (count == 0 && F_if.rst_n)? 1 : 0; // FIX
```

Bugs report summary.	
1. wr_ack	reset when reset is activated
2. overflow	reset when reset is activated
3. underflow	Add instruction to make it sequential.
4. underflow	reset when reset is activated
5. data_out	reset when reset is activated
6. count	Adding case when wr_en and rd_en are high
7. almostfull	High when count less than FIFO_DEPTH by 1 not 2
8. overflow	Must get low when full is zero
9. If statement	Should be “&&” not “&”
10. empty	empty must get low when reset is activated

Questa Snippet:

```
# UVM_INFO UVM_top\Test\test.sv(76) @ 10: uvm_test_top [run_phase] FIFO write only seq
# UVM_INFO UVM_top\Test\test.sv(80) @ 42: uvm_test_top [run_phase] FIFO read only seq
# UVM_INFO UVM_top\Test\test.sv(83) @ 54: uvm_test_top [run_phase] FIFO write and read seq - WRITE DIST MORE THAN READ -
# UVM_INFO UVM_top\Test\test.sv(87) @ 26054: uvm_test_top [run_phase] FIFO sync toggle seq
# UVM_INFO UVM_top\Test\test.sv(90) @ 32052: uvm_test_top [run_phase] FIFO read only seq
# UVM_INFO UVM_top\Test\test.sv(93) @ 32084: uvm_test_top [run_phase] FIFO write only seq
# UVM_INFO UVM_top\Test\test.sv(96) @ 32096: uvm_test_top [run_phase] FIFO async toggle seq
# UVM_INFO UVM_top\Test\test.sv(99) @ 38094: uvm_test_top [run_phase] FIFO read only seq
# UVM_INFO UVM_top\Test\test.sv(102) @ 38126: uvm_test_top [run_phase] FIFO write only seq
# UVM_INFO UVM_top\Test\test.sv(106) @ 38138: uvm_test_top [run_phase] FIFO write and read seq - READ DIST MORE THAN WRITE -
# UVM_INFO UVM_top\Test\test.sv(109) @ 64138: uvm_test_top [run_phase] FIFO reset seq
# UVM_INFO UVM_top\Test\test.sv(112) @ 64148: uvm_test_top [run_phase] FIFO random seq
```

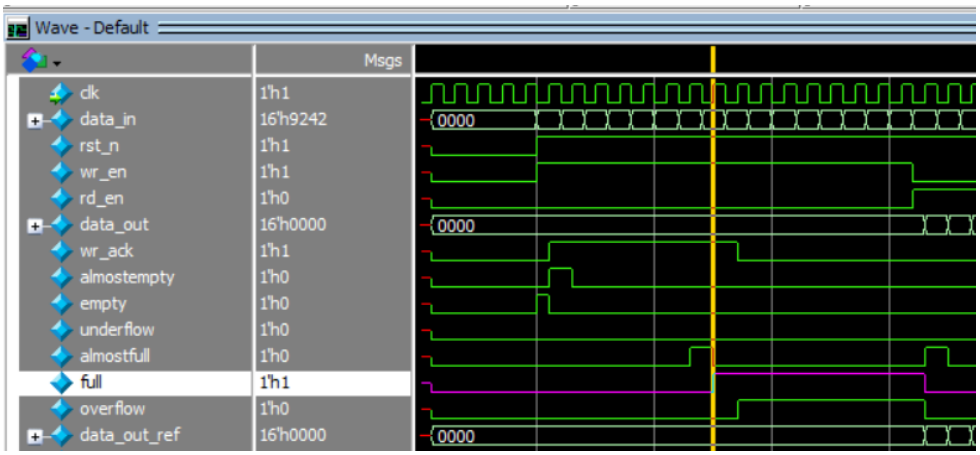
1. Activate reset at the first 5 cycle.



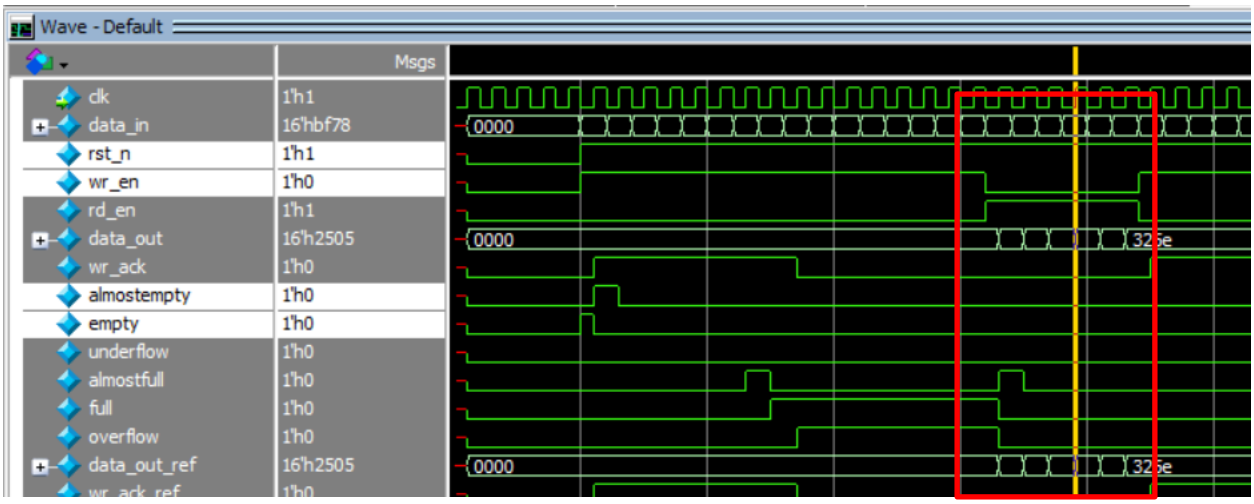
2. LOCK RESET

3. Write only to make FIFO full.

4. Write when FIFO full

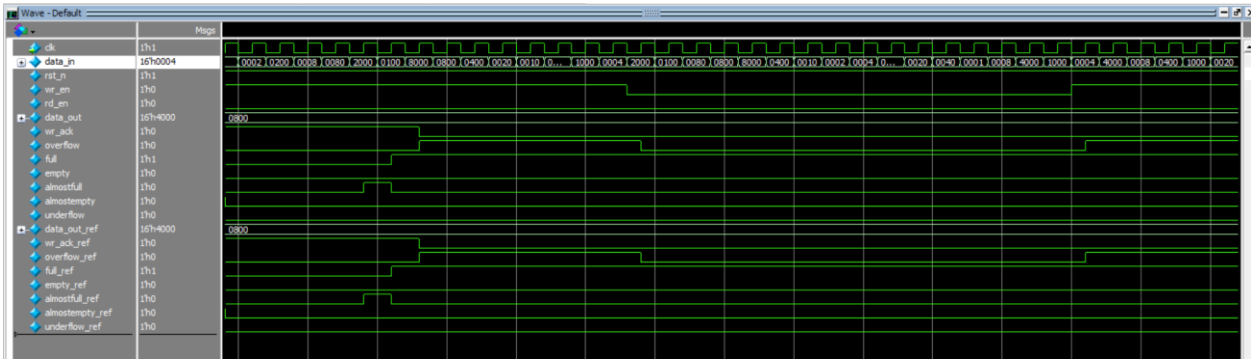


5. Read times less than FIFO_DEPTH.

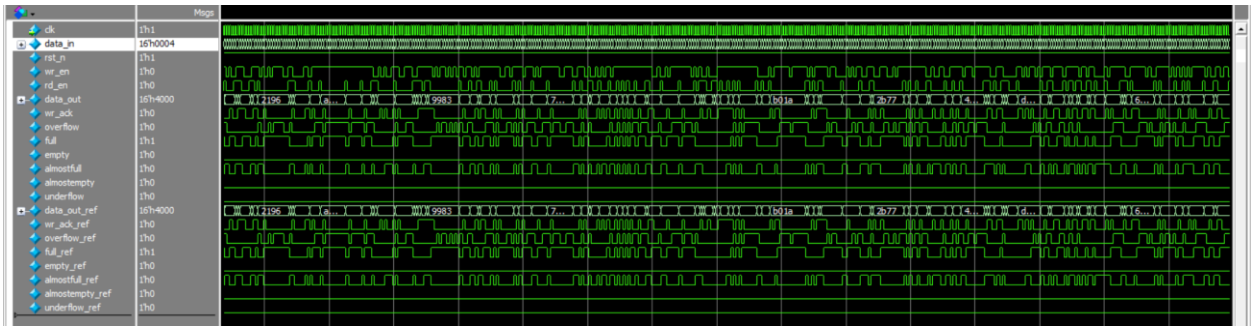


6. read and write constraint `WRITE DIST MORE THAN READ`.

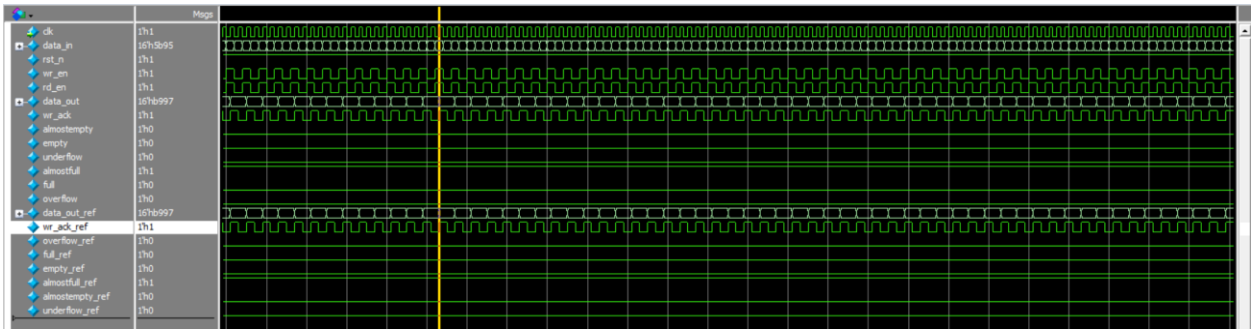
- the first loop to make data_in constraint one bit high



- the seconed loop with no constraint at data_in



7. Apply Sync toggle => read and write will be equal but they are not equal to the same value twice in a row.

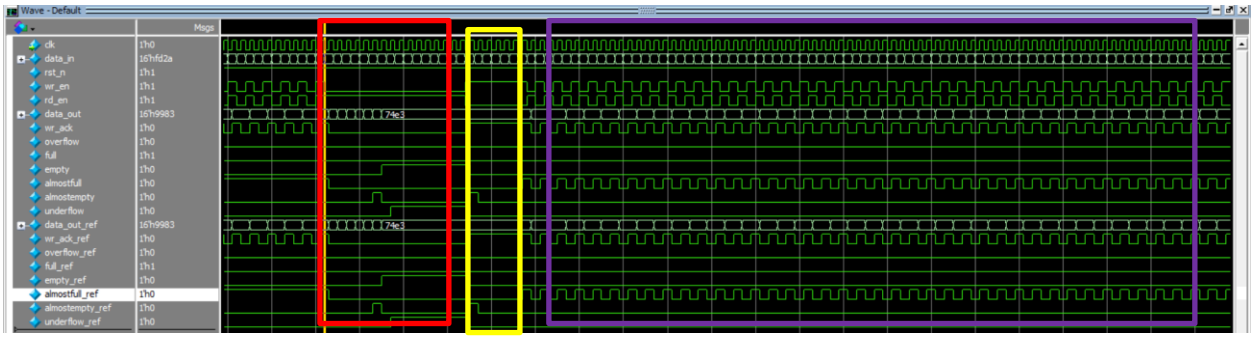


8. Read only to make FIFO empty.

9. Read when FIFO empty.

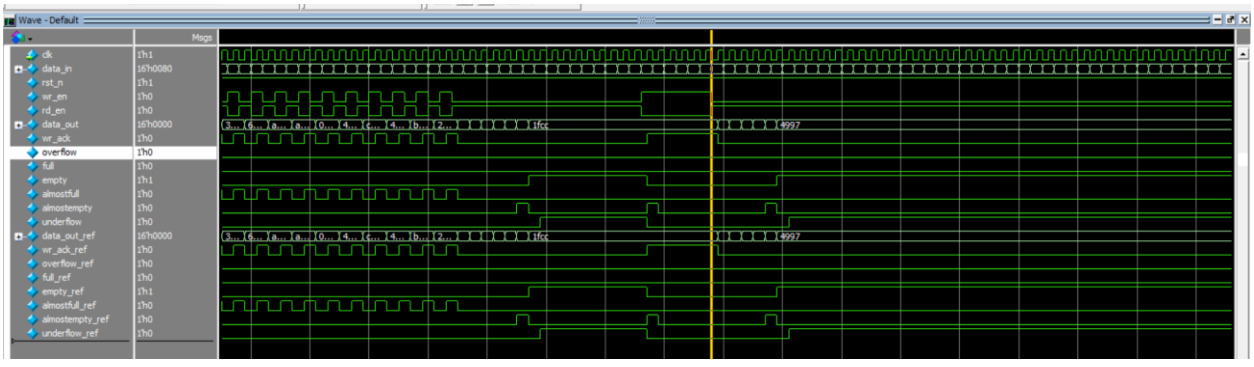
10. **Write only to make FIFO before full.**

11. Apply Async toggle => read and write will be not equal but they are not equal to the same value twice in a row.



12. Read Only.

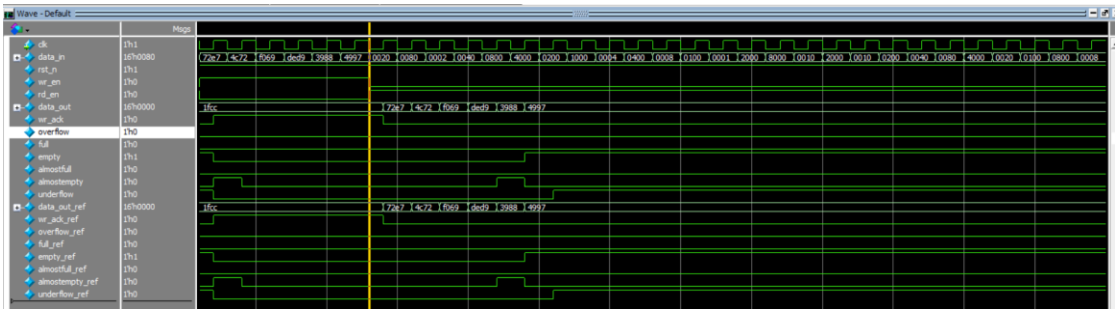
13. write Only.



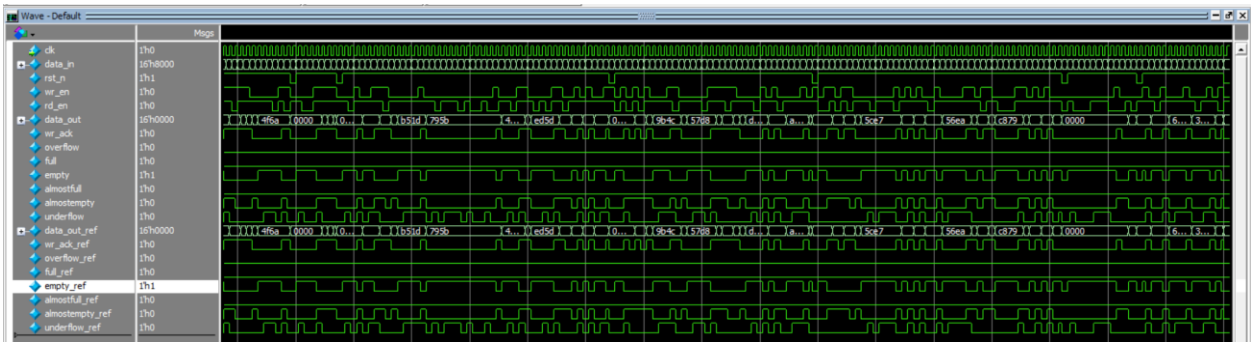
14. UNLOOK RESET

15. Apply read and write at the same time `WRITE DIST LESS THAN READ`.

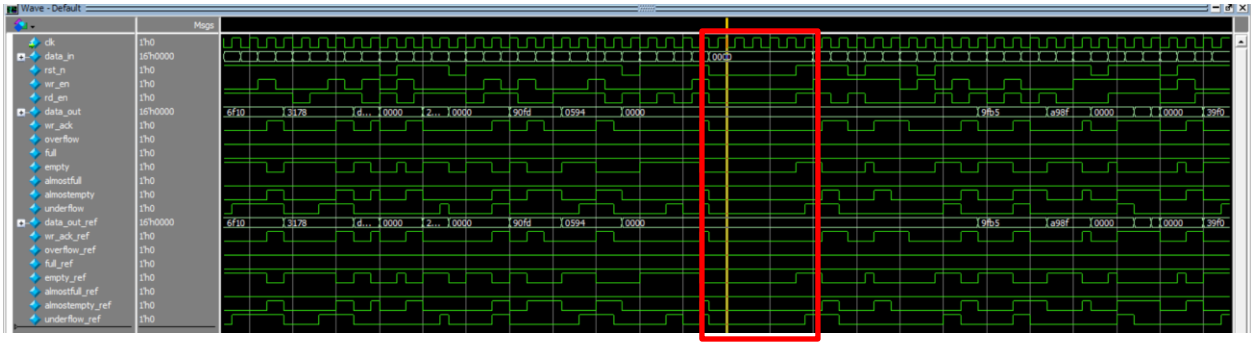
- the first loop to make data_in constraint one bit high



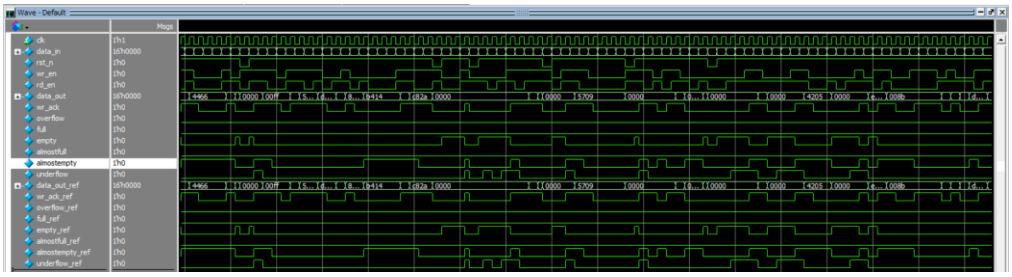
- the seconed loop with no constraint at data_in



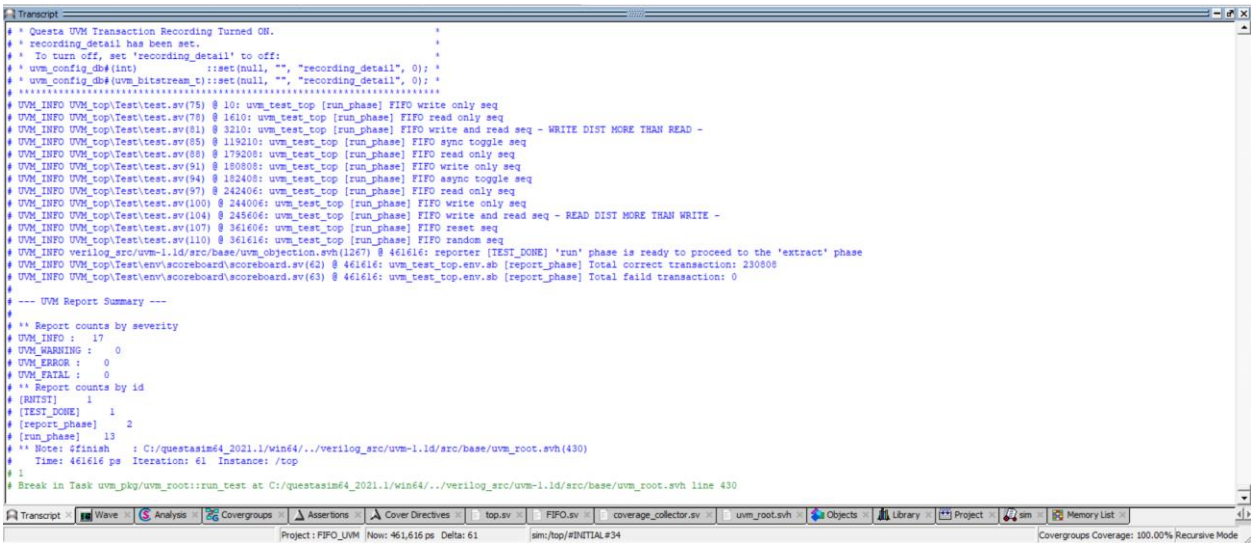
16. Activate reset.



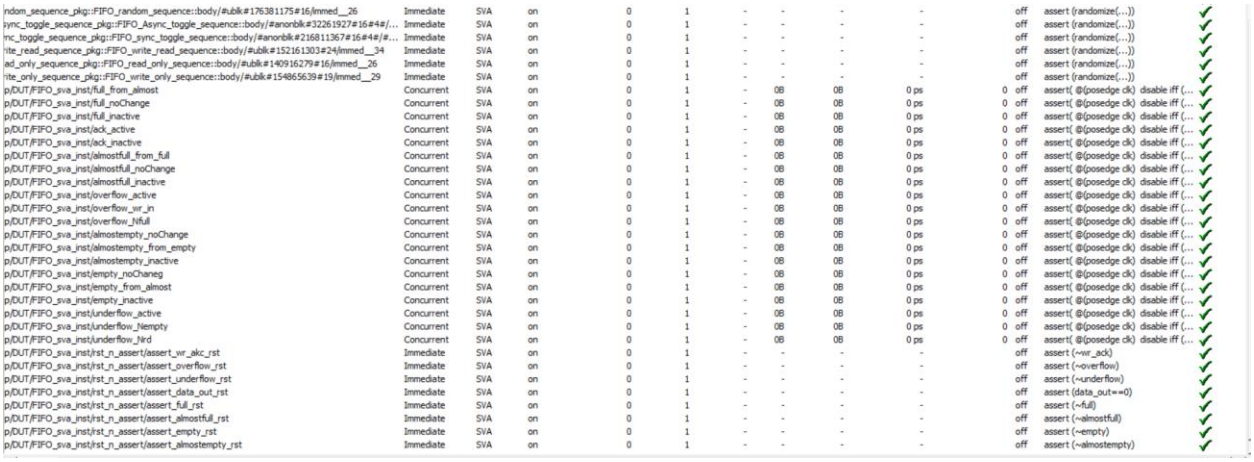
17. Randomization with no Constraint.



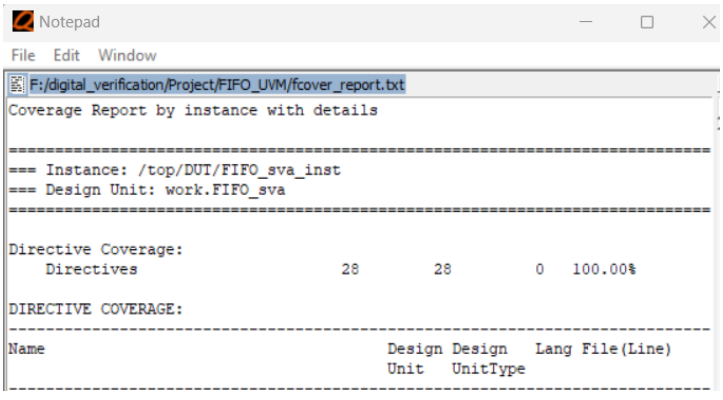
/// Summary at the end of simulation ///



/// Assertion ///

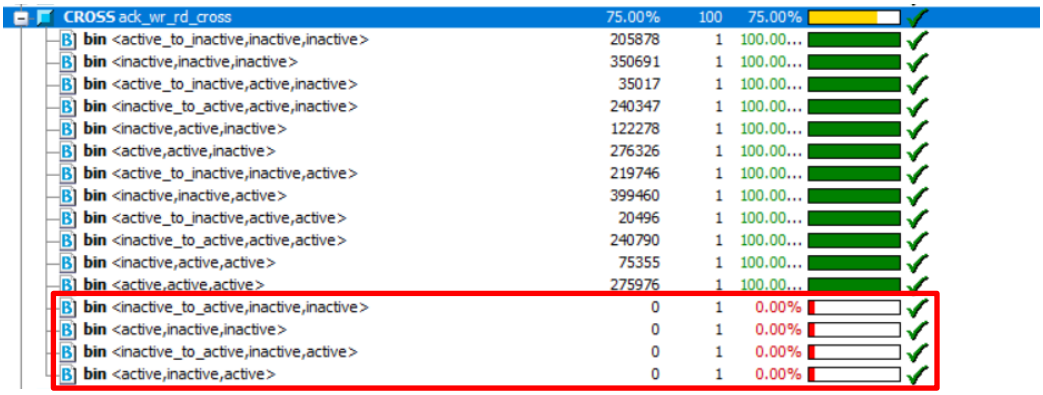


/// Coverage directive ///



There is cross coverage when it's zero, it makes me sure my design behave well Like:

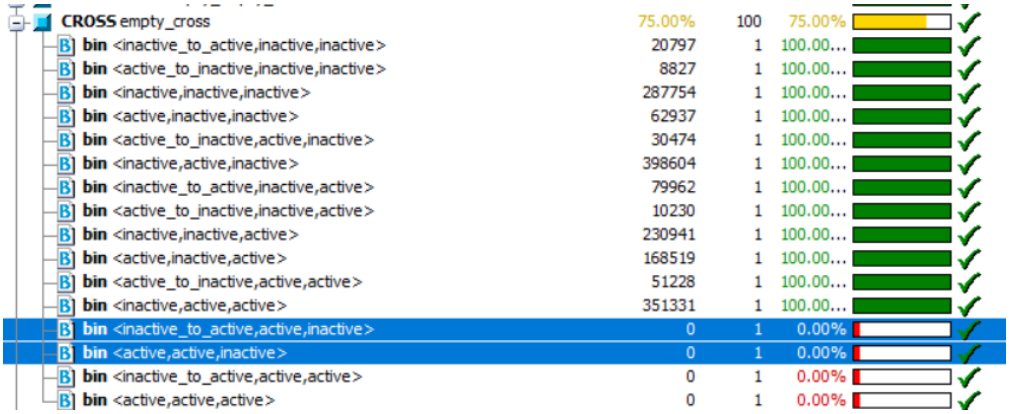
1. wr_ack will not be active in case of wr_en is inactive



CROSS ack_wr_rd_cross	75.00%	100	75.00%
bin <active_to_inactive,inactive,inactive>	205878	1	100.00...
bin <inactive,inactive,inactive>	350691	1	100.00...
bin <active_to_inactive,active,inactive>	35017	1	100.00...
bin <inactive_to_active,active,inactive>	240347	1	100.00...
bin <inactive,active,inactive>	122278	1	100.00...
bin <active,active,inactive>	276326	1	100.00...
bin <active_to_inactive,inactive,active>	219746	1	100.00...
bin <inactive,inactive,active>	399460	1	100.00...
bin <active_to_inactive,active,active>	20496	1	100.00...
bin <inactive_to_active,active,active>	240790	1	100.00...
bin <inactive,active,active>	75355	1	100.00...
bin <active,active,active>	275976	1	100.00...
bin <inactive_to_active,inactive,inactive>	0	1	0.00%
bin <active,inactive,inactive>	0	1	0.00%
bin <inactive_to_active,inactive,active>	0	1	0.00%
bin <active,inactive,active>	0	1	0.00%

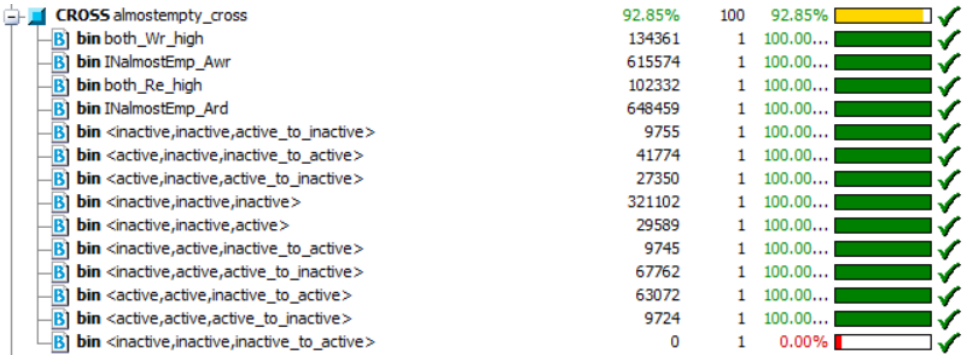
2.empty flag will not activated when rd_en is inactive

```
// rd_en and wr_en
empty_cross: cross empty_cp, wr_en_cp, rd_en_cp;
```



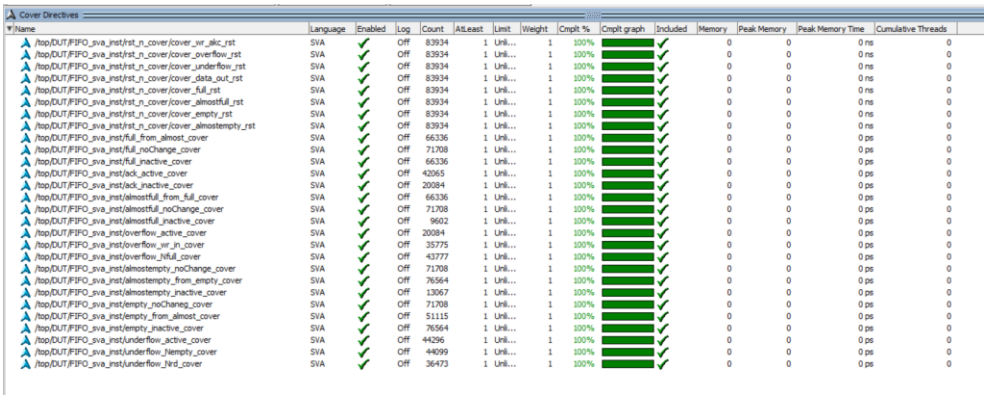
CROSS empty_cross	75.00%	100	75.00%
bin <inactive_to_active,inactive,inactive>	20797	1	100.00...
bin <active_to_inactive,inactive,inactive>	8827	1	100.00...
bin <inactive,inactive,inactive>	287754	1	100.00...
bin <active,inactive,inactive>	62937	1	100.00...
bin <active_to_inactive,active,inactive>	30474	1	100.00...
bin <inactive,active,inactive>	398604	1	100.00...
bin <inactive_to_active,inactive,active>	79962	1	100.00...
bin <active_to_inactive,inactive,active>	10230	1	100.00...
bin <inactive,inactive,active>	230941	1	100.00...
bin <active,inactive,active>	168519	1	100.00...
bin <active_to_inactive,active,active>	51228	1	100.00...
bin <inactive,active,active>	351331	1	100.00...
bin <inactive_to_active,active,inactive>	0	1	0.00%
bin <active,active,inactive>	0	1	0.00%
bin <inactive_to_active,active,active>	0	1	0.00%
bin <active,active,active>	0	1	0.00%

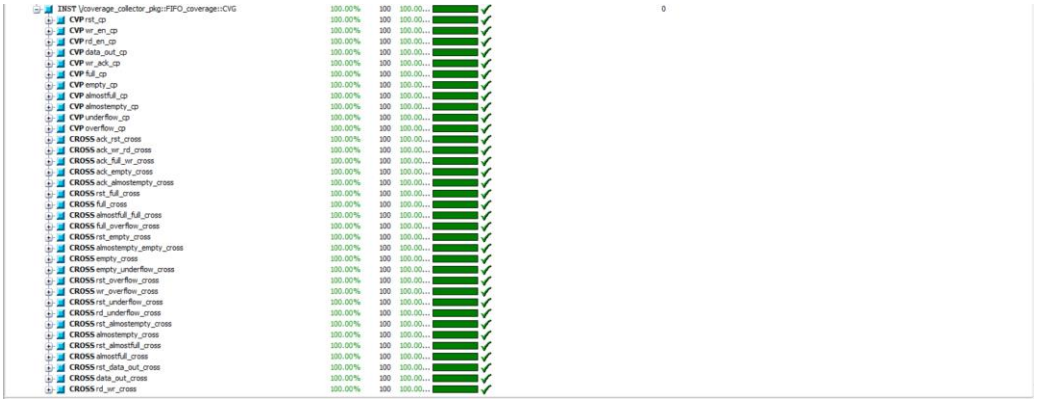
3. almostempty cant trans from inactive to active while read and write is inactive and the same for almostfull



CROSS almostempty_cross	92.85%	100	92.85%
bin both_Wr_high	134361	1	100.00...
bin INalmostEmp_Awr	615574	1	100.00...
bin both_Re_high	102332	1	100.00...
bin INalmostEmp_Ard	648459	1	100.00...
bin <inactive,inactive,active_to_inactive>	9755	1	100.00...
bin <active,inactive,inactive_to_active>	41774	1	100.00...
bin <active,inactive,active_to_inactive>	27350	1	100.00...
bin <inactive,inactive,inactive>	321102	1	100.00...
bin <inactive,inactive,active>	29589	1	100.00...
bin <inactive,active,inactive_to_active>	9745	1	100.00...
bin <inactive,active,active_to_inactive>	67762	1	100.00...
bin <active,active,inactive_to_active>	63072	1	100.00...
bin <active,active,active_to_inactive>	9724	1	100.00...
bin <inactive,inactive,inactive_to_active>	0	1	0.00%

/// Coverage ///






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

Covergroup Coverage:
    Covergroups                     1        na        na   100.00%
        Coverpoints/Crosses        35        na        na        na
            Covergroup Bins       171       171         0   100.00%
```

```
 ================================================================================
 === Instance: /top/DUT/FIFO_sva_inst
 === Design Unit: work.FIFO_sva
 ================================================================================

 Directive Coverage:
     Directives                      28        28         0   100.00%
```

/// Code Coverage Summary ///

Instance ↑	Branches	Conditions	Directives	Statements	Toggles	Total
Search...	Search...	Search...	Search...	Search...	Search...	Search...
Total	100%	100%	100%	100%	100%	100%
top	-	-	-	100%	100%	100%
DUT	100%	100%	100%	100%	100%	100%
GLD	100%	100%	-	100%	100%	100%
intf	-	-	-	-	100%	100%

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

```
 Condition Coverage:
     Enabled Coverage              Bins   Covered    Misses  Coverage
     ----------------              ----      ----    ------  --------
     Conditions                      25        25         0   100.00%

 =================================Condition Details=================================
```

```
 Statement Coverage:
     Enabled Coverage              Bins      Hits    Misses  Coverage
     ----------------              ----      ----    ------  --------
     Statements                      30        30         0   100.00%

 ================================Statement Details================================
```

FIFO_interfase toggle for all signals (inputs and outputs)

```
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                        106       106         0   100.00%

================================Toggle Details================================
```

FIFO_sva

```
================================================================================
=== Instance: /top/DUT/FIFO_sva_inst
=== Design Unit: work.FIFO_sva
================================================================================
Branch Coverage:
    Enabled Coverage             Bins      Hits    Misses  Coverage
    ----------------             ----      ----    ------  --------
    Branches                        4         4         0   100.00%
```

```
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         86        86         0   100.00%

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

Toggle Coverage for instance /top/DUT/FIFO_sva_inst --
```