**UVM project - Synchronous FIFO**

**Abdelrahman Mohamed Ali**

**7 May, 2024**

|  |  |
| --- | --- |
| **Gmail** | [**abdelrahmansalby23@gmail.com**](mailto:abdelrahmansalby23@gmail.com) |
| **LinkedIn** | [**Abdelrahman Mohamed**](https://www.linkedin.com/in/abdelrahman-mohamed-814a9022a/) |
| **GitHub** | [**Abdelrahman1810**](https://github.com/Abdelrahman1810?tab=repositories) |

**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  4.checked data\_out  5.Count will not change | 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).  data\_out&wr\_en&rd\_en (data\_out\_cross). | Output  Checked against golden model  Checked by assertion Label:  **ack\_active,**  **count\_noChange** |
| 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**

Write only

Read only

write only

Read probability higher than Write

Read only

Write only

**sync. Toggle**

Read only

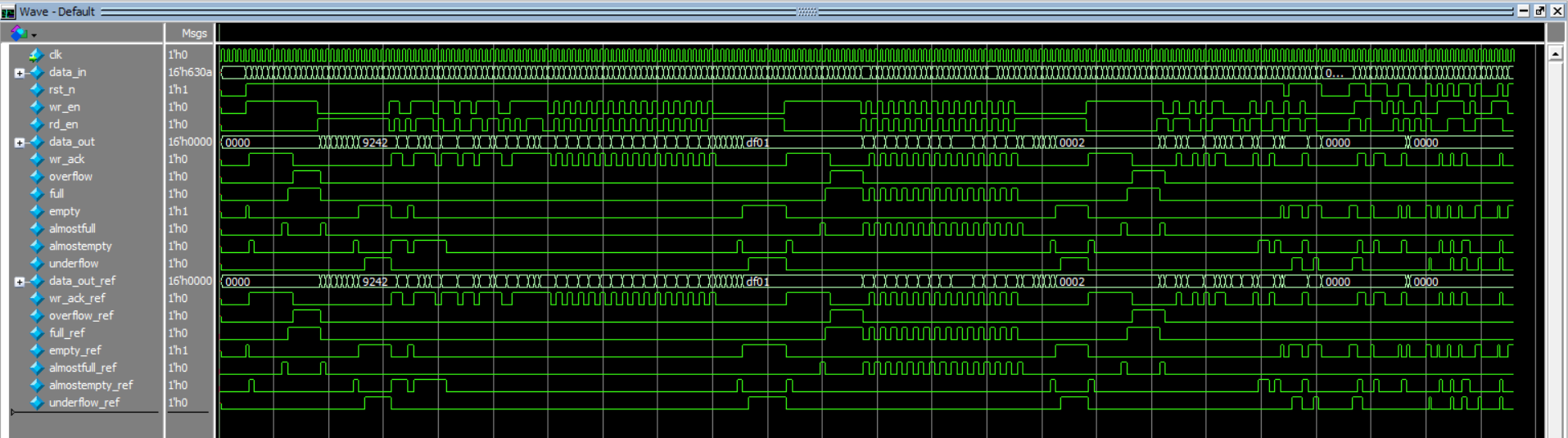
Write probability higher than Read

**rst**

**rst**

**No constraint**

**Async. Toggle**

****

**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 {Codes\shared\_pkg\shared\_pkg.sv}

###

vlog -coveropt 3 +cover +acc {Codes\Interface\interface.sv}

vlog -coveropt 3 +cover +acc {Codes\refrence\FIFO\_ref.sv}

###

vlog -coveropt 3 +cover +acc {Codes\DUT\FIFO.sv}

vlog +define+LOOK\_ASSERTION -coveropt 3 +cover +acc {DUT\assertion.sv}

###

vlog -coveropt 3 +cover +acc {Codes\objects\configration.sv}

#vlog -coveropt 3 +cover +acc {Codes\objects\sequence\_Items\sequenceItem\_Valid.sv}

vlog -coveropt 3 +cover +acc {Codes\objects\sequence\_Item\sequenceItem.sv}

vlog -coveropt 3 +cover +acc {Codes\objects\FIFO\_sequence\FIFO\_reset\_sequence.sv}

vlog -coveropt 3 +cover +acc {Codes\objects\FIFO\_sequence\FIFO\_write\_only\_sequence.sv}

vlog -coveropt 3 +cover +acc {Codes\objects\FIFO\_sequence\FIFO\_read\_only\_sequence.sv}

vlog -coveropt 3 +cover +acc {Codes\objects\FIFO\_sequence\FIFO\_write\_read\_sequence.sv}

vlog -coveropt 3 +cover +acc {Codes\objects\FIFO\_sequence\FIFO\_sync\_toggle\_sequence.sv}

vlog -coveropt 3 +cover +acc {Codes\objects\FIFO\_sequence\FIFO\_Async\_toggle\_sequence.sv}

vlog -coveropt 3 +cover +acc {Codes\objects\FIFO\_sequence\FIFO\_random\_sequence.sv}

###

vlog -coveropt 3 +cover +acc {Codes\UVM\_top\Test\env\agent\driver\driver.sv}

vlog -coveropt 3 +cover +acc {Codes\UVM\_top\Test\env\agent\monitor\monitor.sv}

vlog -coveropt 3 +cover +acc {Codes\UVM\_top\Test\env\agent\sequencer\sequencer.sv}

vlog -coveropt 3 +cover +acc {Codes\UVM\_top\Test\env\agent\agent.sv}

###

vlog -coveropt 3 +cover +acc {Codes\UVM\_top\Test\env\coverage\_collector\coverage\_collector.sv}

vlog -coveropt 3 +cover +acc {Codes\UVM\_top\Test\env\scoreboard\scoreboard.sv}

vlog -coveropt 3 +cover +acc {Codes\UVM\_top\Test\env\env.sv}

###

vlog -coveropt 3 +cover +acc {Codes\UVM\_top\Test\test.sv}

vlog -coveropt 3 +cover +acc {Codes\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

/// Defines

`define OFF\_ALL item.constraint\_mode(0);

`define ON(constraint) item.constraint.constraint\_mode(1);

`define create\_comp(type, name) type::type\_id::create(name, this);

`define create\_obj(type, name) type::type\_id::create(name);

`define asrt\_prp(content) assert property (@(posedge clk) disable iff(!rst\_n) (content));

`define cov\_prp(content) cover property (@(posedge clk) disable iff(!rst\_n) (content));

`define asrt\_fn assert final

`define cov\_fn cover final

`define same\_seq (rd\_en && wr\_en && !empty && !full)

`define Func\_new\_p(name\_) \

function new(string name = name\_, uvm\_component parent = null); \

    super.new(name, parent); \

endfunction //new()

`define Func\_new(name\_) \

function new(string name = name\_); \

    super.new(name); \

endfunction //new()

/// 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;

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);              ||

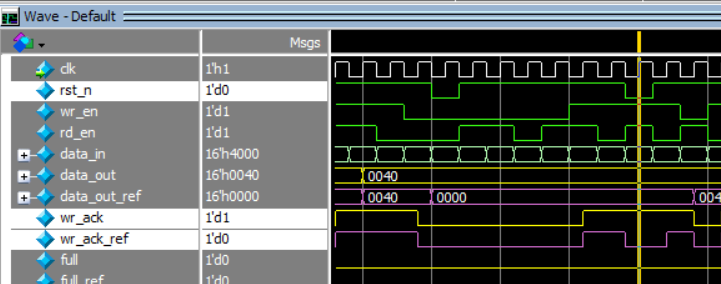
|| last.clk.cycle.underflow=1 & rd\_en=0, overflow deactivated ||underflow\_Nrd:     `asrt\_prp(($past(underflow) && !rd\_en)  |=> !underflow);||

|| 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 \*\* ///**





FIX:

    if (!F\_if.rst\_n) begin

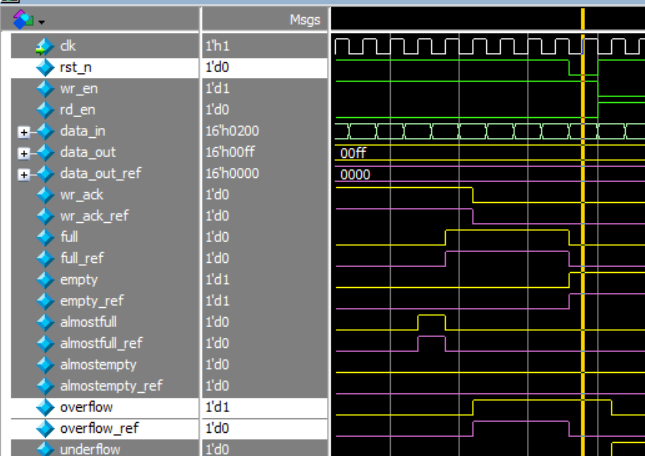
        wr\_ptr <= 0;

        F\_if.wr\_ack <= 0; //

    end

**/// \*\* Bug\_2: The signal overflow is sequential so it must be reset to zero when reset is activate \*\* ///**





Fix:

    if (!F\_if.rst\_n) begin

        wr\_ptr <= 0;

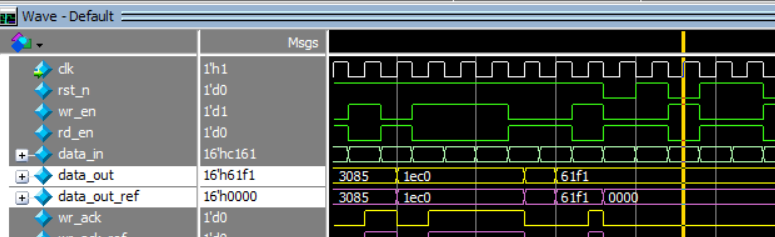
        F\_if.wr\_ack <= 0; //

        F\_if.overflow <= 0; //

    end

**/// \*\* Bug\_3: The data\_out bus is sequential so it must be reset to zero when reset is activate \*\* ///**





FIX:

    if (!F\_if.rst\_n) begin

        rd\_ptr <= 0;

        F\_if.underflow <= 0;// Fix

        F\_if.data\_out <= 0;// Fix

    end

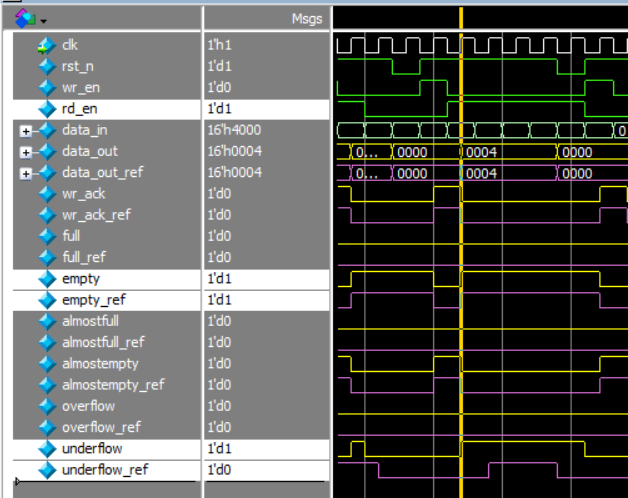
**/// \*\* Bug\_4: underflow must be sequential not combinational \*\* ///**

Bug in code:

assign underflow = (empty && rd\_en)? 1 : 0;

Bug in waveform:





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)

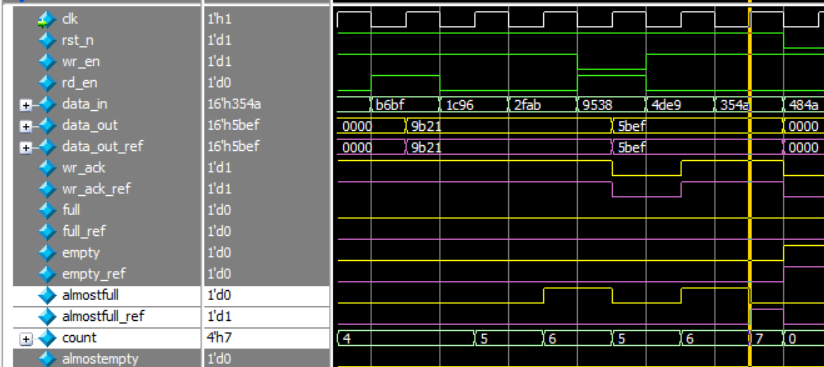
            count <= count + 1;

        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\*\* ///**





assign almostfull = (count == FIFO\_DEPTH-2)? 1 : 0;

Fix:

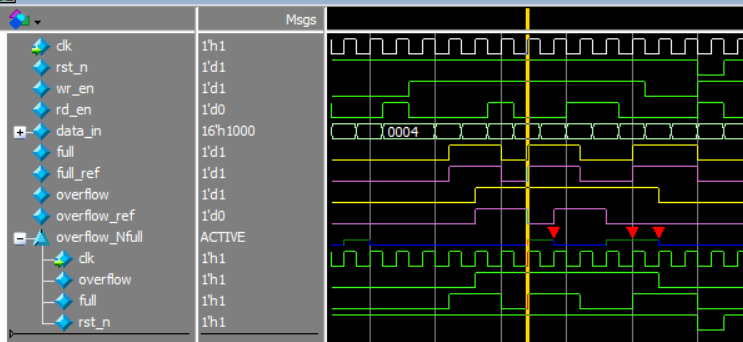
assign F\_if.almostfull = (count == FIFO\_DEPTH-1)? 1 : 0; //

**/// \*\* 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



****

Fix:

    else if (F\_if.wr\_en && count < FIFO\_DEPTH) begin

        mem[wr\_ptr] <= F\_if.data\_in;

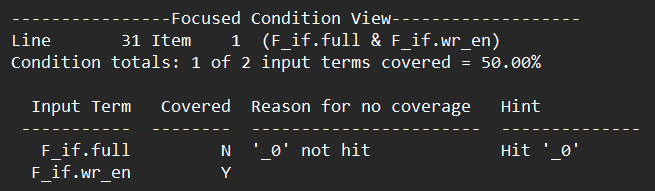
        F\_if.wr\_ack <= 1;

        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\*\* ///**

****

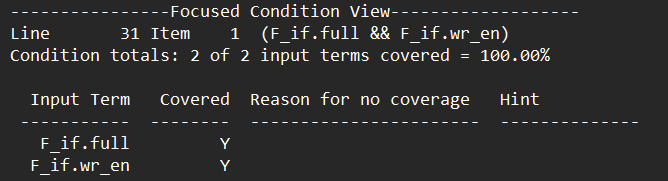
        if (F\_if.full & F\_if.wr\_en)

            F\_if.overflow <= 1;

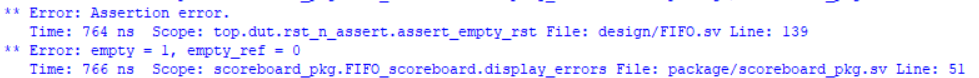
Fix:

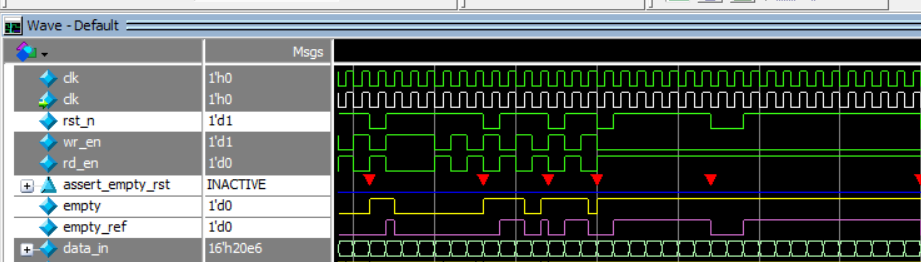
        if (F\_if.full && F\_if.wr\_en)// FIX

            F\_if.overflow <= 1;



**/// \*\* Bug\_9: when rst\_n is activated then empty signal must be zero\*\* ///**

****

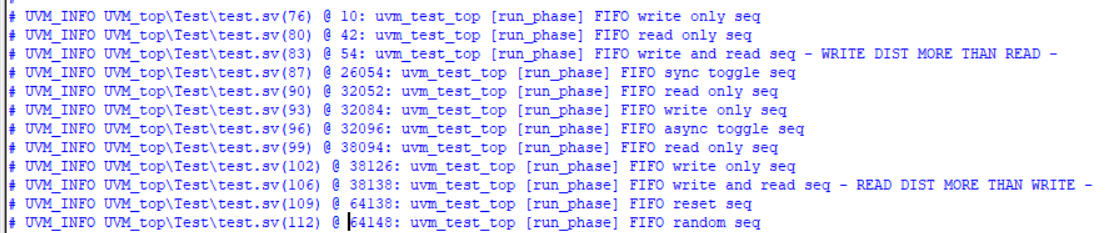
****

Fix:

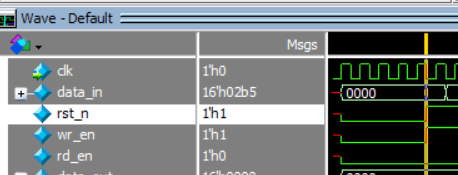
assign F\_if.empty = (count == 0 && F\_if.rst\_n)? 1 : 0; // FIX

|  |  |
| --- | --- |
| Bugs report summary. | |
| 1. wr\_ack | reset when reset is activated |
| 1. overflow | reset when reset is activated |
| 1. underflow | Add instruction to make it sequential. |
| 1. underflow | reset when reset is activated |
| 1. data\_out | reset when reset is activated |
| 1. count | Adding case when wr\_en and rd\_en are high |
| 1. almostfull | High when count less than FIFO\_DEPTH by 1 not 2 |
| 1. overflow | Must get low when full is zero |
| 1. If statement | Should be “&&” not “&” |
| 1. empty | empty must get low when reset is activated |

**Questa Snippet:**



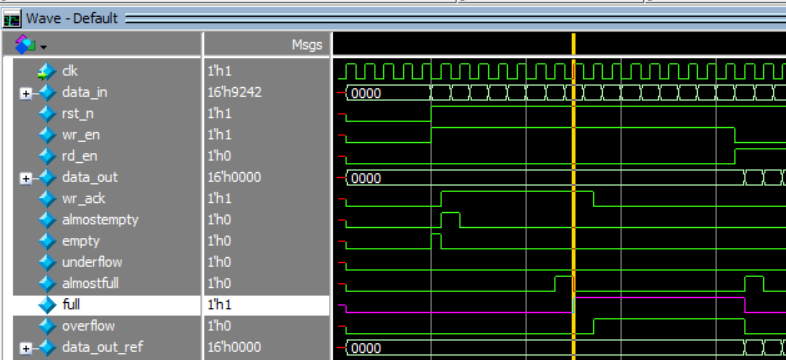
1. Activate reset at the first 5 cycle.



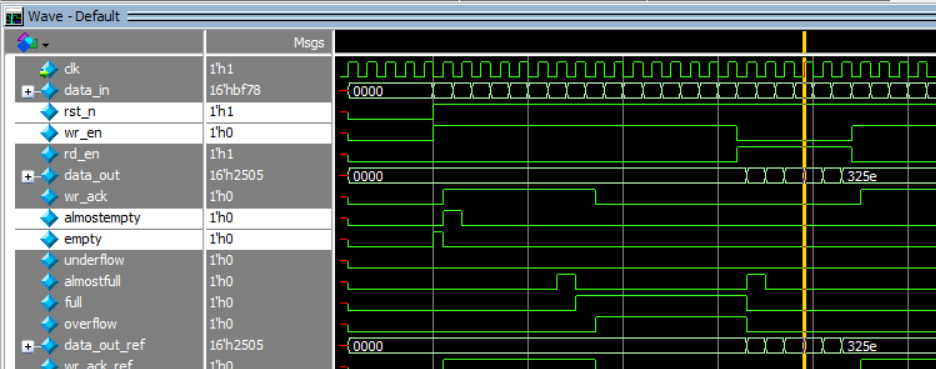
2. LOCK RESET

3. Write only to make FIFO full.

4. Write when FIFO full

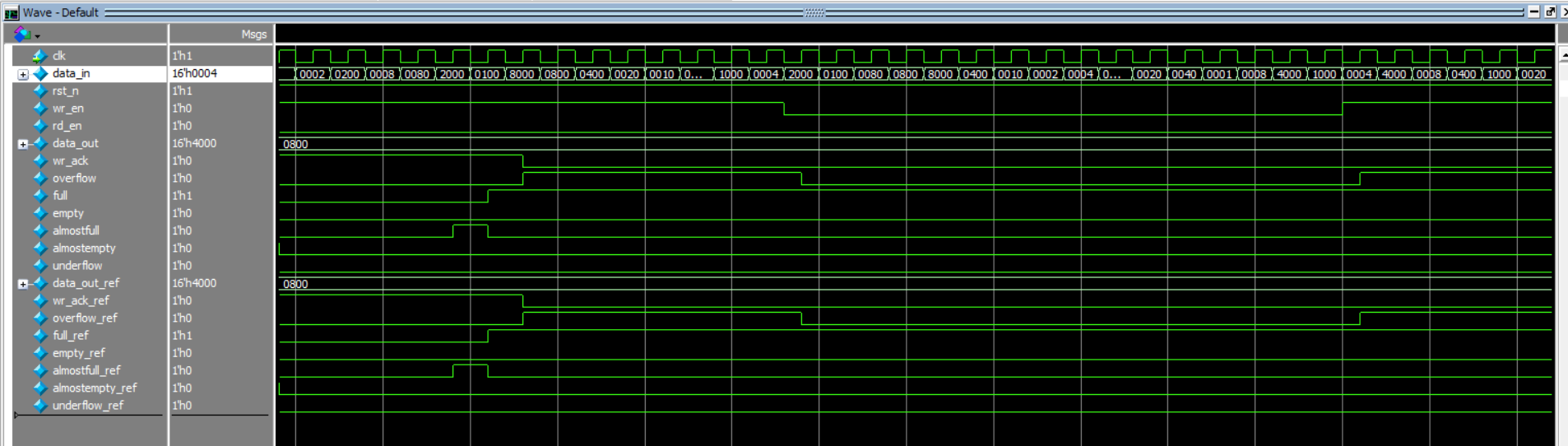


5. Read times less than FIFO\_DEPTH.

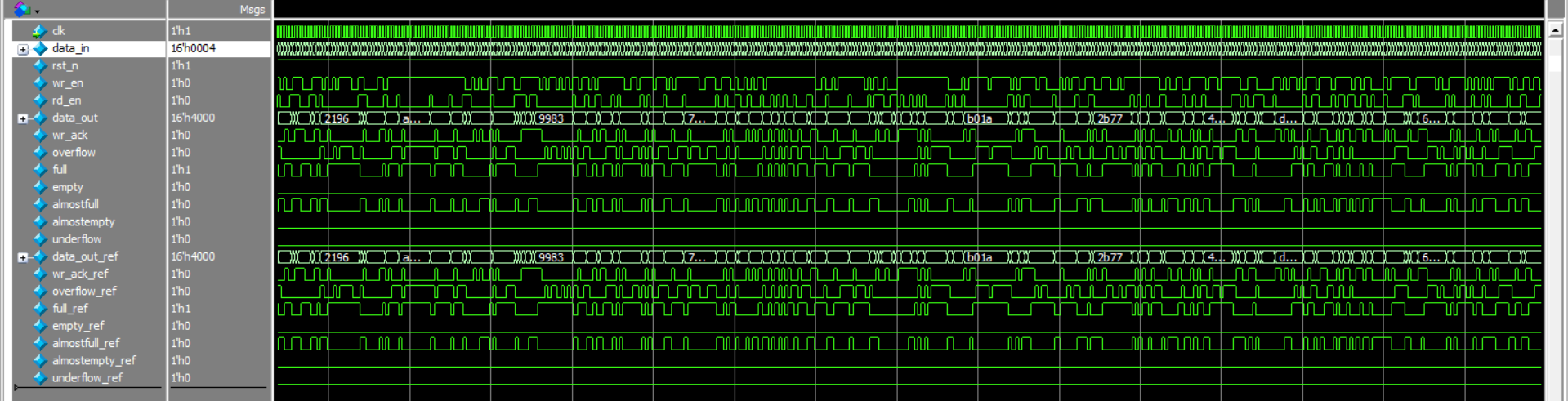


6. read and write constraint `WRITE DIST MORE THAN READ`.

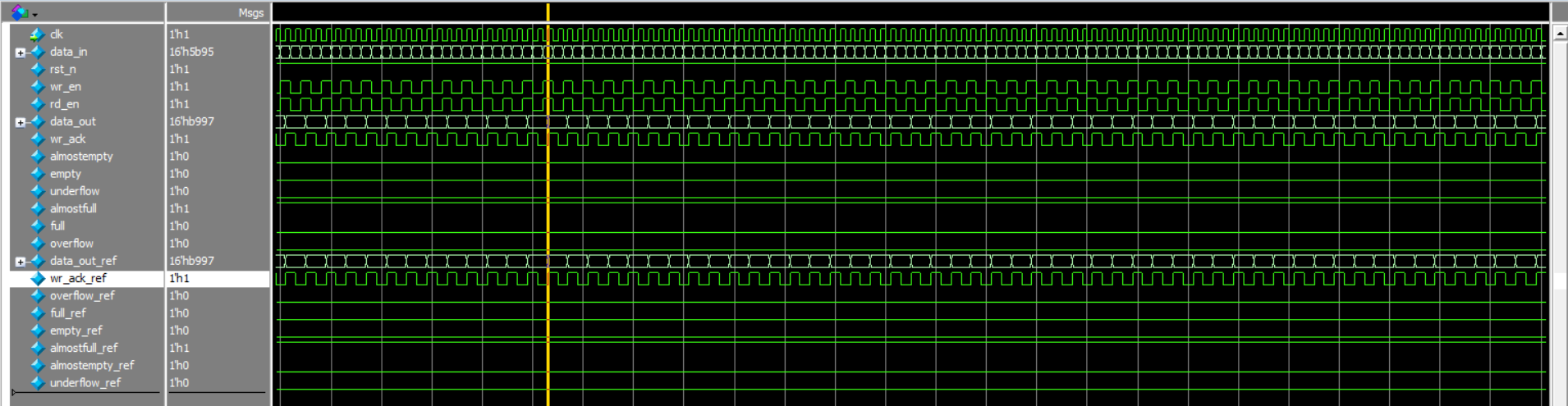
- the first loop to make data\_in constraint one bit high



- 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.

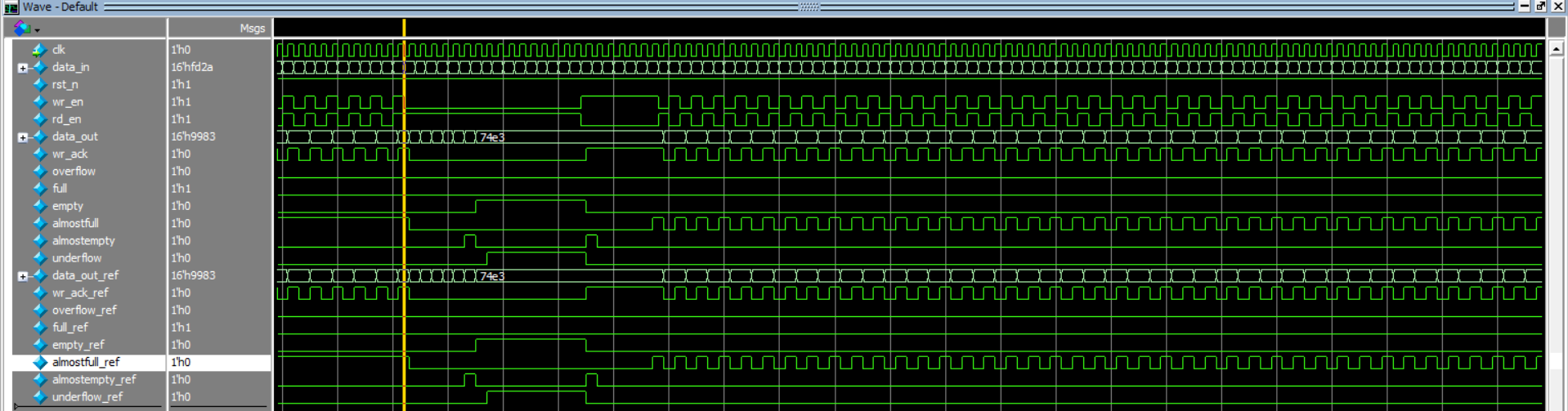


8. Read only to make FIFO empty.

9. Read when FIFO empty.

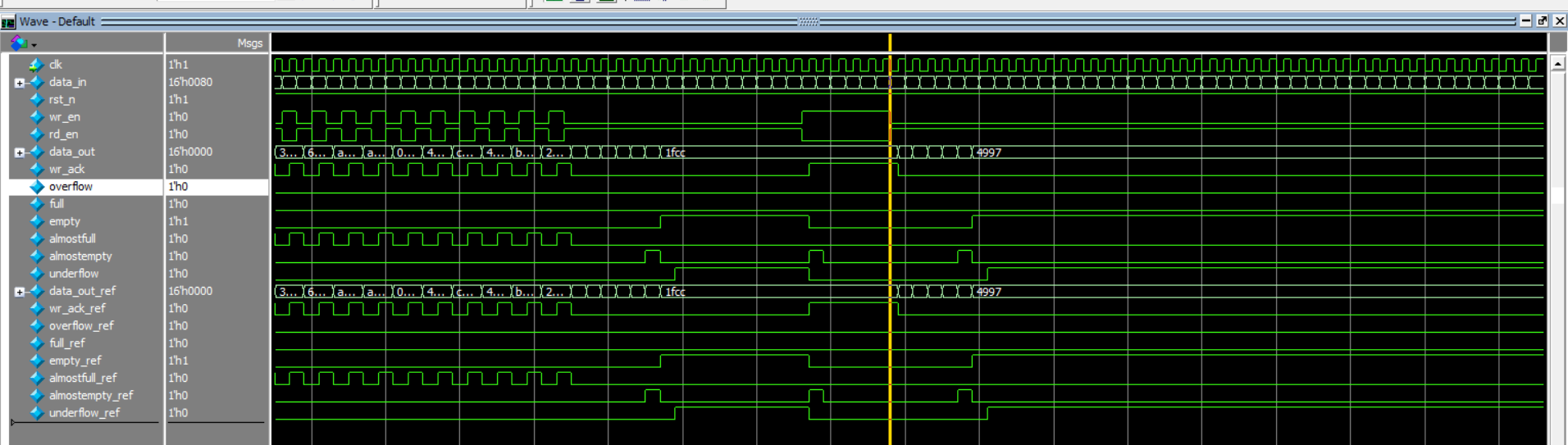
10. **Write only to make FIFO before full.**

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.

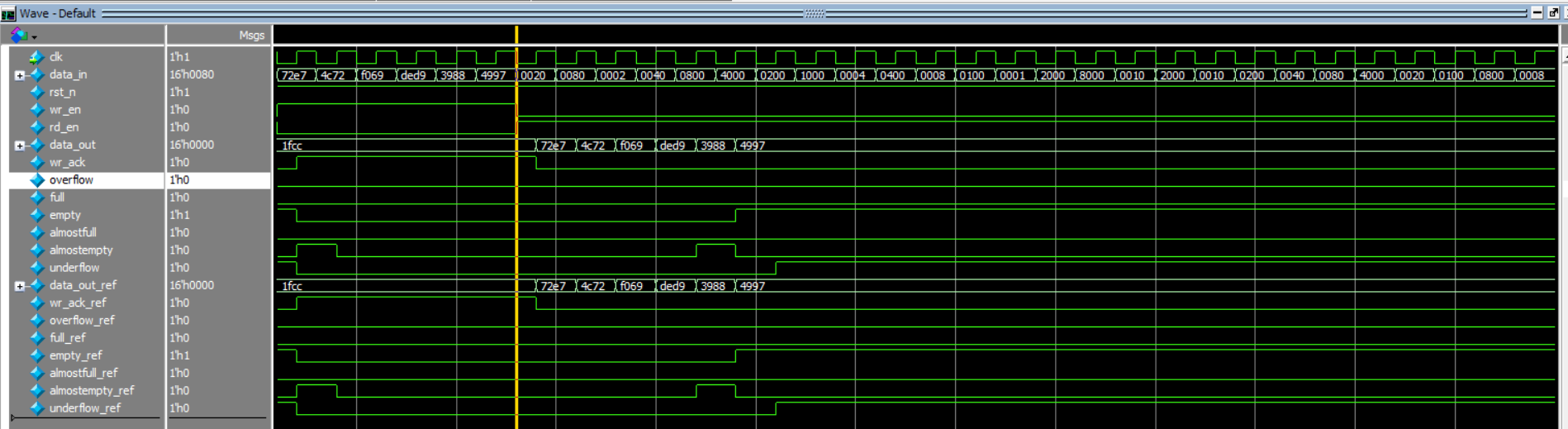
13. write Only.



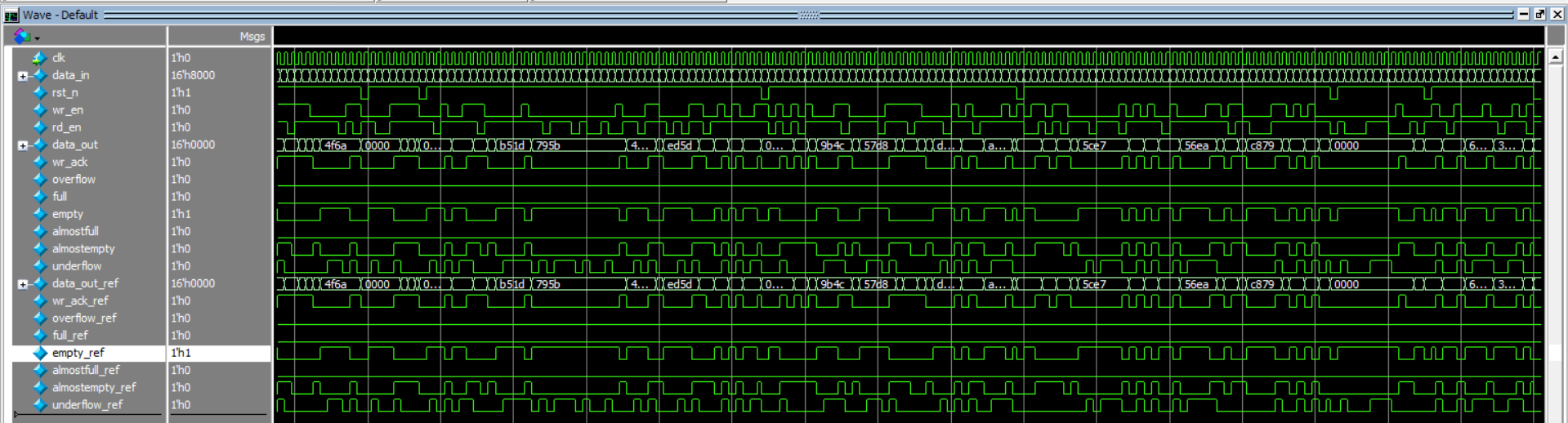
14. UNLOOK RESET

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

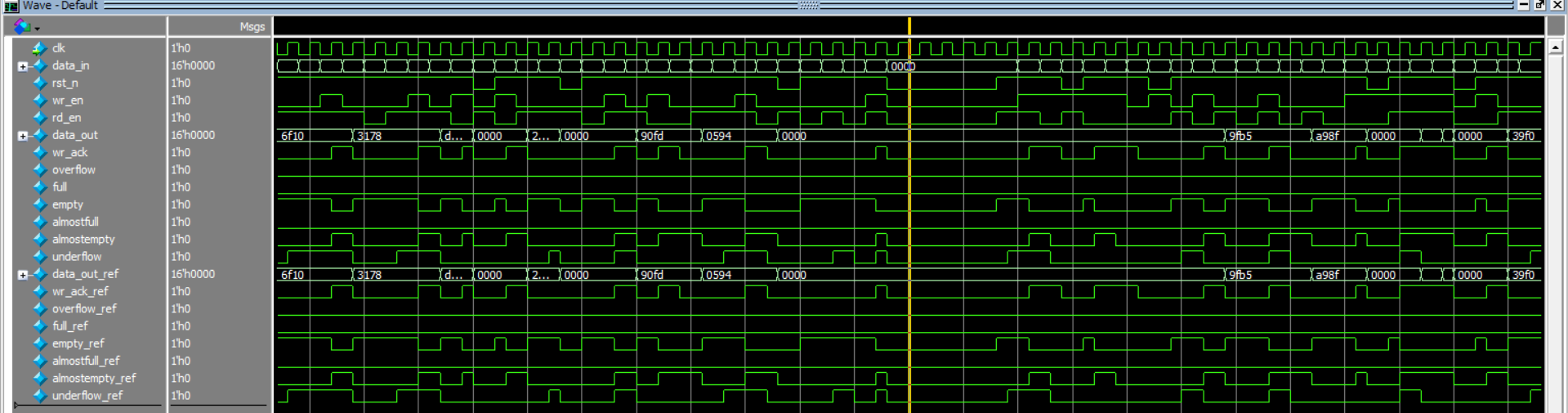
- the first loop to make data\_in constraint one bit high



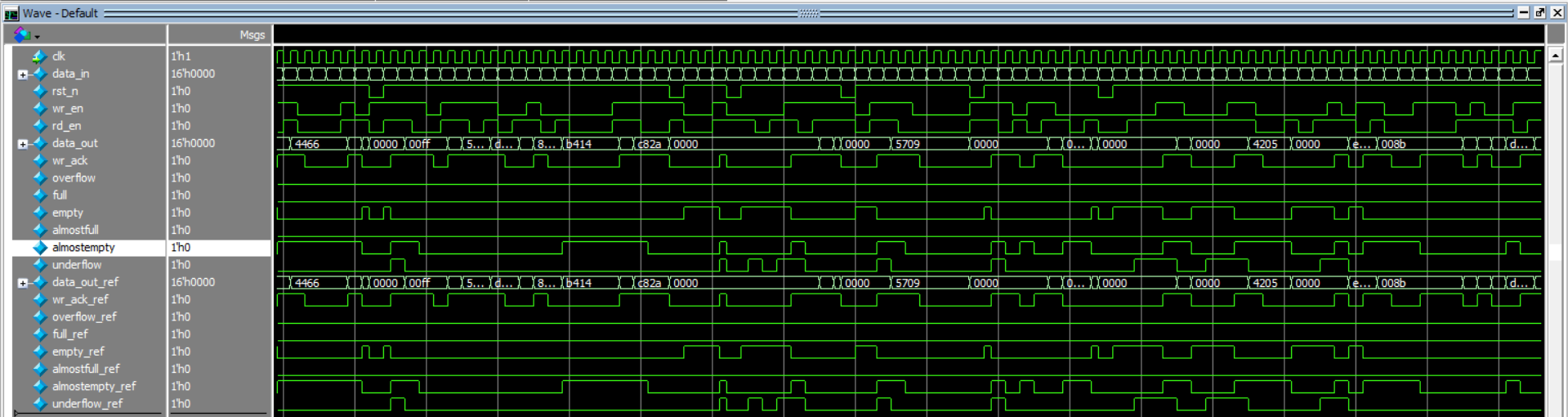
- the seconed loop with no constraint at data\_in



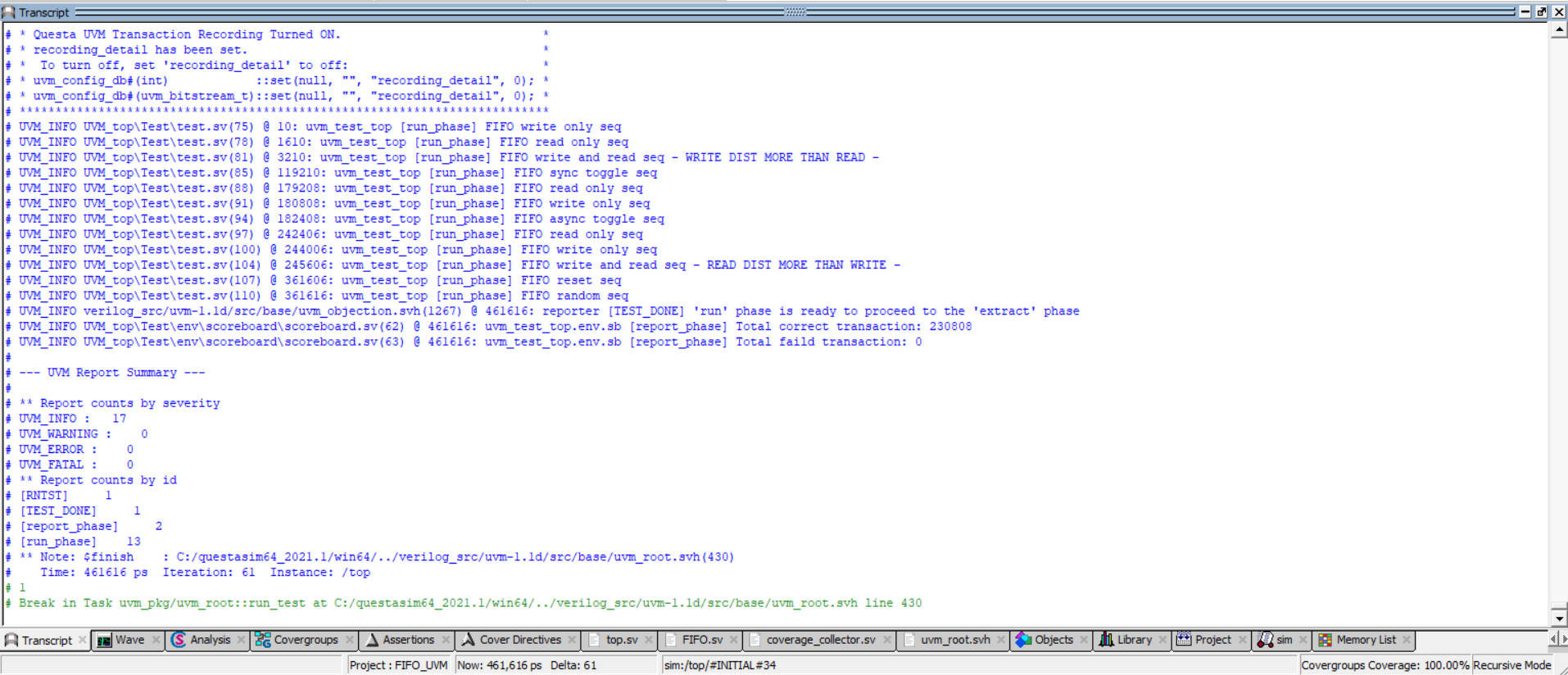
16. Activate reset.



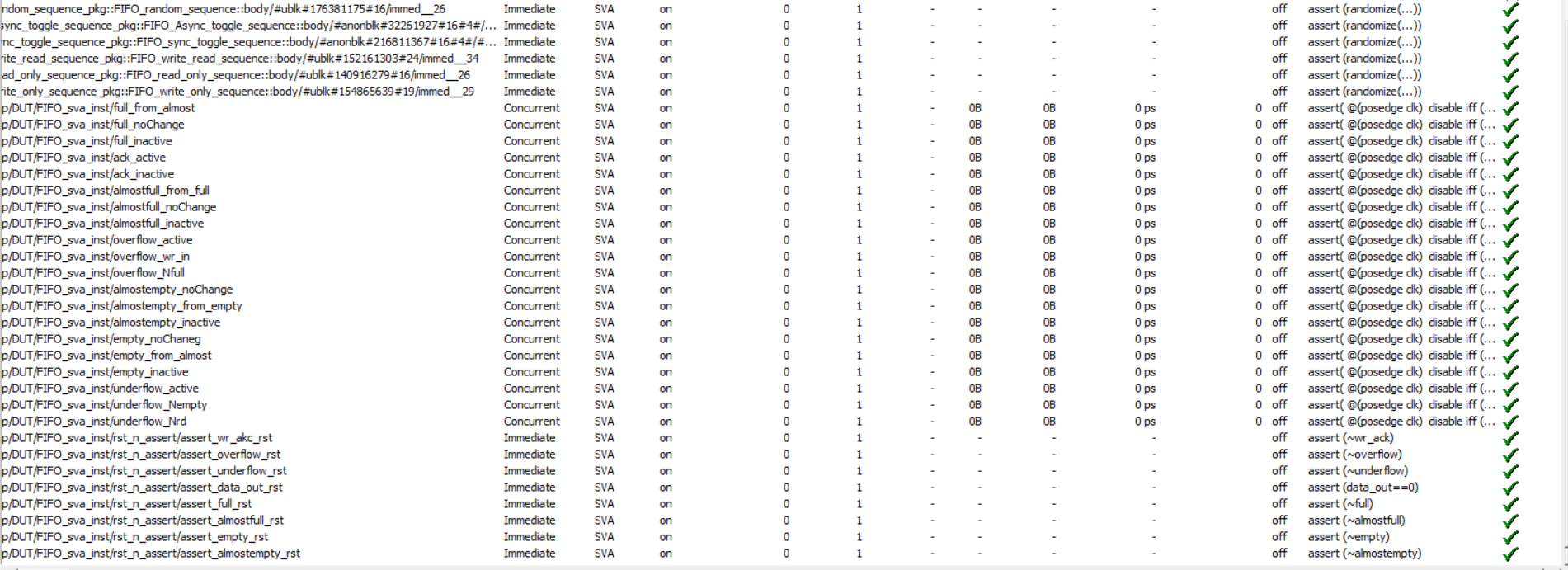
17. Randomization with no Constraint.



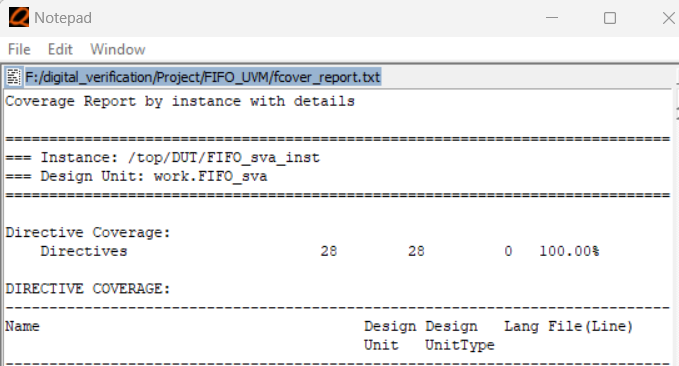
**/// Summary at the end of simulation ///**



**/// Assertion ///**

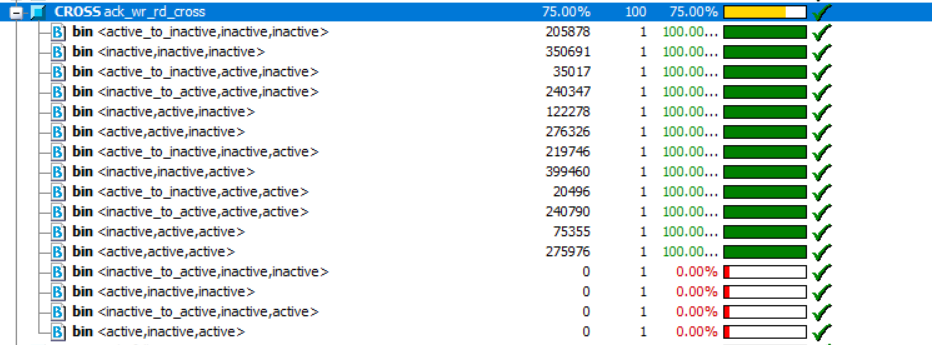


**/// Coverage directive ///**

****

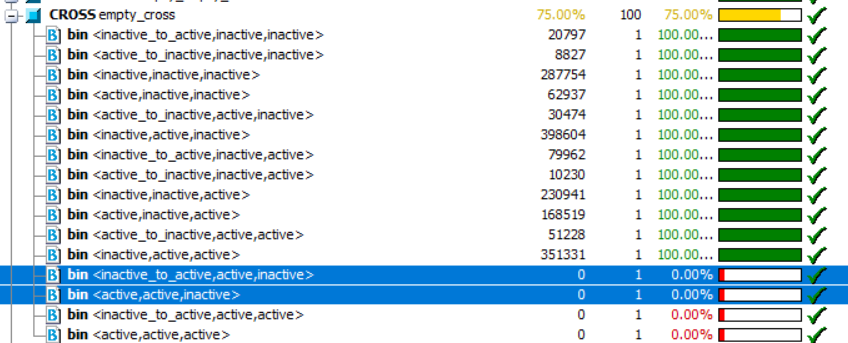
**There is cross coverage when it’s zero, it makes me sure my design behave well Like:**

**1. wr\_ack will not be active in case of wr\_en is inactive**

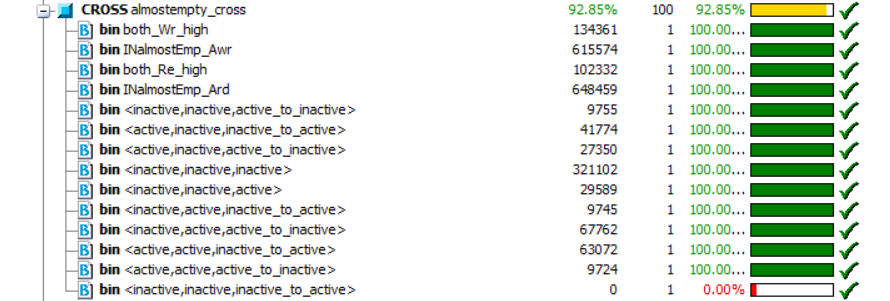
****

**2.empty flag will not activated when rd\_en is inactive**

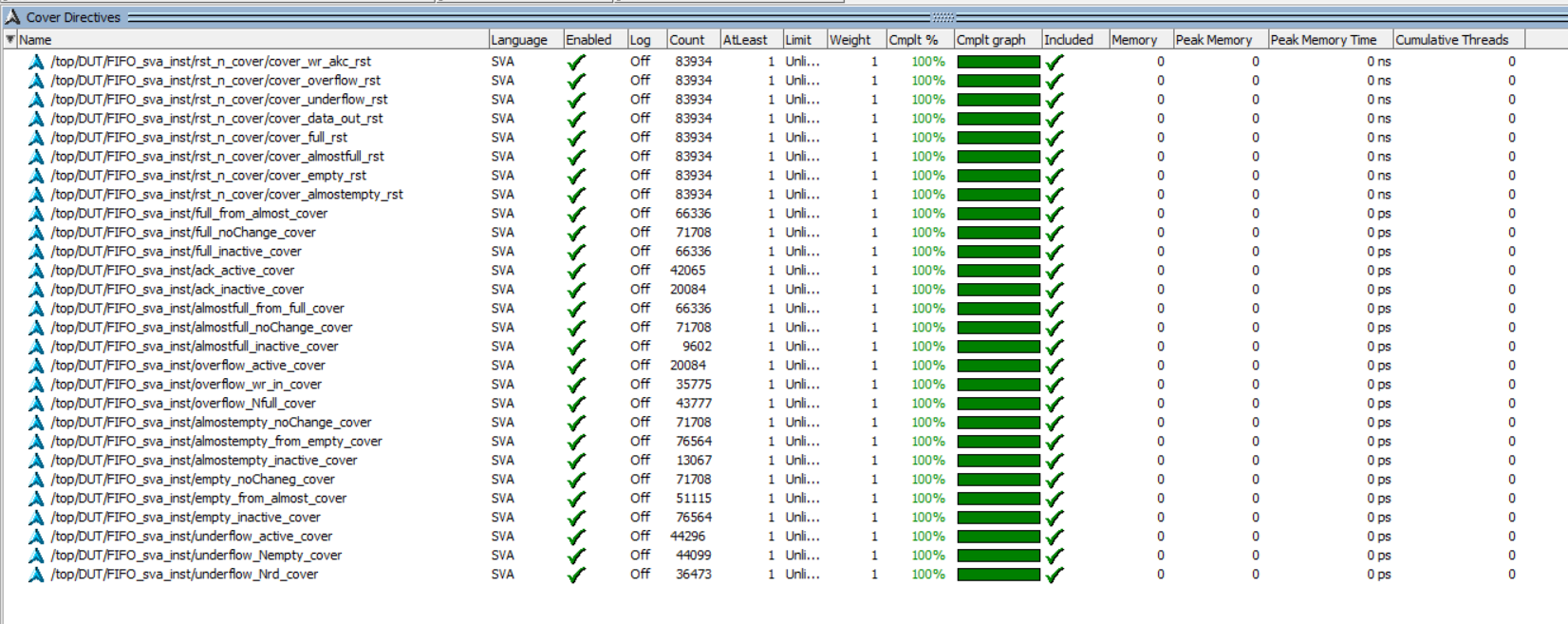
****

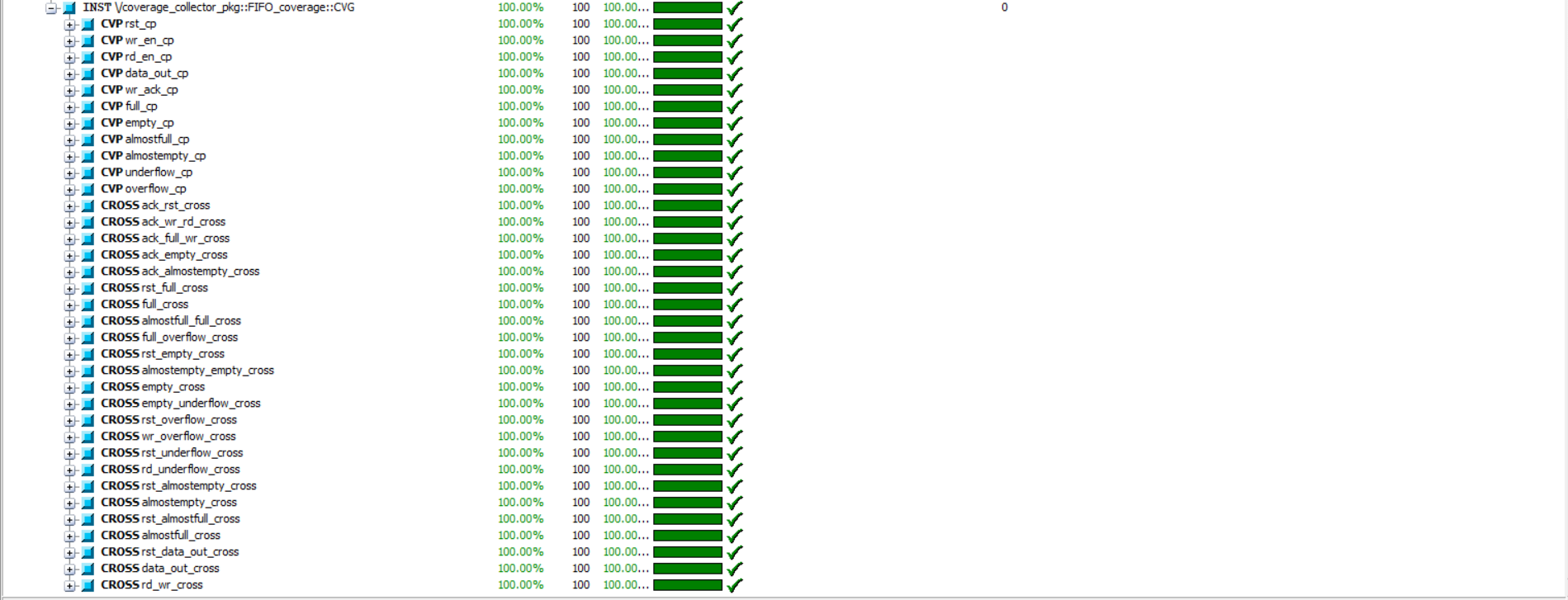
****

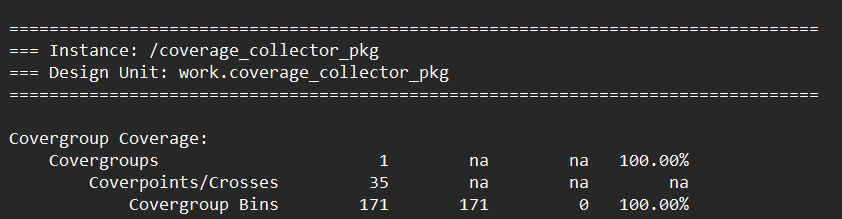
**3. almostempty cant trans from inactive to active while read and write is inactive and the same for almostfull**

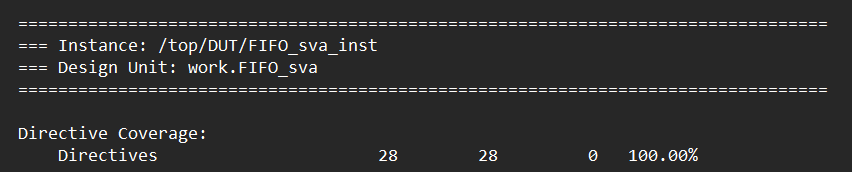
****

**/// Coverage ///**

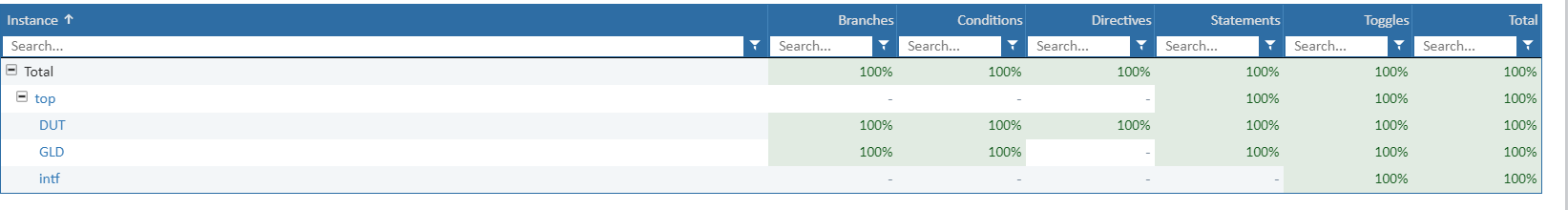
****

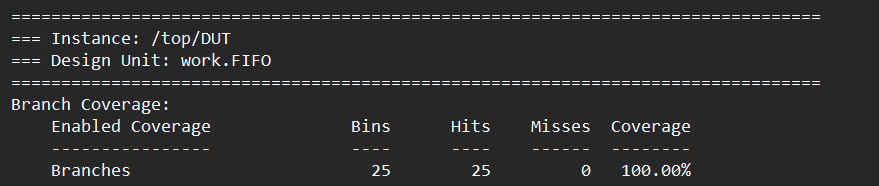


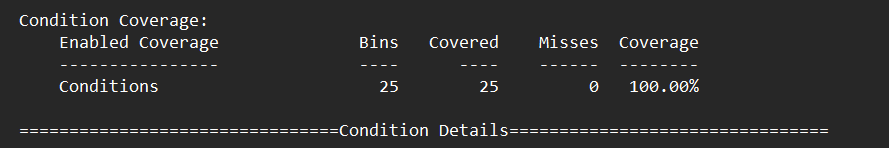


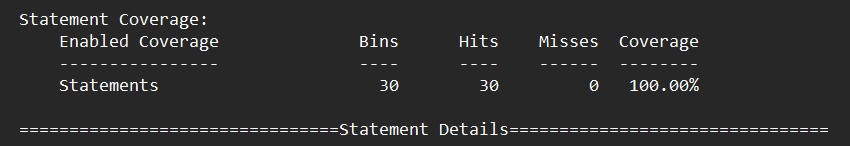


**/// Code Coverage Summary ///**

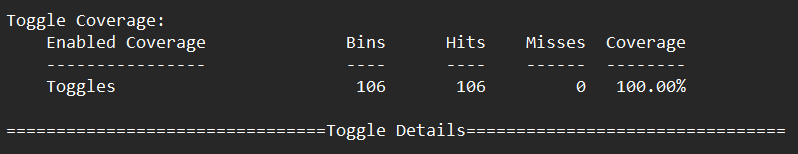








FIFO\_interfase toggle for all signals (inputs and outputs)



FIFO\_sva

