**SV project - Synchronous FIFO**

**Abdelrahman Mohamed Ali**

**30 April, 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) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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: MY\_CON\_ONLEY\_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: MY\_CON\_ONLEY\_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: MY\_CON\_ONLEY\_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: MY\_CON\_ONLEY\_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: MY\_CON\_ONLEY\_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: MY\_CON\_ONLEY\_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: MY\_CON\_ONLEY\_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: MY\_CON\_ONLEY\_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: MY\_CON\_ONLEY\_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: MY\_CON\_ONLEY\_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: MY\_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: MY\_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: MY\_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: MY\_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: MY\_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** |
|  |  |  |  |  |

**/// Do file**

vlib work

vlog -coveropt 3 +cover +acc {Codes\package\shared\_pkg.sv}

vlog -coveropt 3 +cover +acc {Codes\package\transaction\_pkg.sv}

vlog -coveropt 3 +cover +acc {Codes\package\coverage\_pkg.sv}

vlog -coveropt 3 +cover +acc {Codes\package\scoreboard\_pkg.sv}

vlog -coveropt 3 +cover +acc {Codes\interface\FIFO\_interface.SV}

# adding "+define+SIM" to enable assertion if not then don't forget to comment <add wave <assertion label>>

vlog +define+SIM -coveropt 3 +cover +acc {Codes\design\FIFO.sv}

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

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

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

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

# this module to test the refrence model

#vlog -coveropt 3 +cover +acc {Codes\testbench\ref\_tb.sv}

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

vsim -voptargs=+acc work.top -cover

add wave \*

add wave -position 1 -color white sim:/top/F\_if/clk

add wave -position 2 -radix unsigned sim:/top/F\_if/rst\_n

add wave -position 3 -radix unsigned sim:/top/F\_if/wr\_en

add wave -position 4 -radix unsigned sim:/top/F\_if/rd\_en

add wave -position 5 -radix hexadecimal sim:/top/F\_if/data\_in

add wave -position 6  -color yellow -radix hexadecimal sim:/top/F\_if/data\_out

add wave -position 7  -color Orchid -radix hexadecimal sim:/top/F\_if/data\_out\_ref

add wave -position 8  -color yellow -radix unsigned sim:/top/F\_if/wr\_ack

add wave -position 9  -color Orchid -radix unsigned sim:/top/F\_if/wr\_ack\_ref

add wave -position 12 -color yellow -radix unsigned sim:/top/F\_if/full

add wave -position 13 -color Orchid -radix unsigned sim:/top/F\_if/full\_ref

add wave -position 14 -color yellow -radix unsigned sim:/top/F\_if/empty

add wave -position 15 -color Orchid -radix unsigned sim:/top/F\_if/empty\_ref

add wave -position 16 -color yellow -radix unsigned sim:/top/F\_if/almostfull

add wave -position 17 -color Orchid -radix unsigned sim:/top/F\_if/almostfull\_ref

add wave -position 18 -color yellow -radix unsigned sim:/top/F\_if/almostempty

add wave -position 19 -color Orchid -radix unsigned sim:/top/F\_if/almostempty\_ref

add wave -position 20 -color yellow -radix unsigned sim:/top/F\_if/overflow

add wave -position 21 -color Orchid -radix unsigned sim:/top/F\_if/overflow\_ref

add wave -position 22 -color yellow -radix unsigned sim:/top/F\_if/underflow

add wave -position 23 -color Orchid -radix unsigned sim:/top/F\_if/underflow\_ref

add wave -position 24 -radix unsigned sim:/top/dut/count

.vcop Action toggleleafnames

## Assertion

add wave /top/dut/rst\_n\_assert/assert\_full\_rst

add wave /top/dut/rst\_n\_assert/assert\_almostfull\_rst

add wave /top/dut/rst\_n\_assert/assert\_empty\_rst

add wave /top/dut/rst\_n\_assert/assert\_almostempty\_rst

add wave /top/dut/full\_count

add wave /top/dut/full\_from\_almost

add wave /top/dut/full\_noChange

add wave /top/dut/full\_inactive

add wave /top/dut/ack\_active

add wave /top/dut/ack\_inactive

add wave /top/dut/almostfull\_count

add wave /top/dut/almostfull\_from\_full

add wave /top/dut/almostfull\_noChange

add wave /top/dut/almostfull\_inactive

add wave /top/dut/overflow\_active

add wave /top/dut/overflow\_wr\_in

add wave /top/dut/overflow\_Nfull

add wave /top/dut/almostempty\_count

add wave /top/dut/almostempty\_noChange

add wave /top/dut/almostempty\_from\_empty

add wave /top/dut/almostempty\_inactive

add wave /top/dut/empty\_count

add wave /top/dut/empty\_noChaneg

add wave /top/dut/empty\_from\_almost

add wave /top/dut/empty\_inactive

add wave /top/dut/underflow\_active

add wave /top/dut/underflow\_Nempty

add wave /top/dut/underflow\_Nrd

add wave /top/dut/count\_inc

add wave /top/dut/count\_dec

add wave /top/dut/count\_noChange

add wave /top/dut/inc\_wr\_ptr\_assert

add wave /top/dut/inc\_rd\_ptr\_assert

## reset assertion

add wave /top/dut/rst\_n\_assert/assert\_count\_rst

add wave /top/dut/rst\_n\_assert/assert\_wr\_ptr\_rst

add wave /top/dut/rst\_n\_assert/assert\_rd\_ptr\_rst

add wave /top/dut/rst\_n\_assert/assert\_wr\_akc\_rst

add wave /top/dut/rst\_n\_assert/assert\_overflow\_rst

add wave /top/dut/rst\_n\_assert/assert\_underflow\_rst

add wave /top/dut/rst\_n\_assert/assert\_data\_out\_rst

run -all

vsim -coverage -vopt work.top -c -do "coverage save -onexit -du FIFO -directive -codeAll cover.ucdb; run -all"

coverage report -detail -cvg -directive -comments -output {Reports/Coverage group report/COV\_GRP\_FIFO.txt} {}

# if you want to see waveform you have top comment "quit -sim" instruction

quit -sim

vcover report cover.ucdb -details -all -annotate -output {Reports/code coverage report/CODE\_COVER\_FIFO.txt}

vcover report -html cover.ucdb -output {Reports/code\_cover\_report\_html/.}

/// Interface

interface FIFO\_interface (clk);

import shared\_pkg::\*;

    input bit clk;

    reg [FIFO\_WIDTH-1:0] data\_in;

    reg rst\_n, wr\_en, rd\_en;

    reg [FIFO\_WIDTH-1:0] data\_out;

    reg wr\_ack, overflow;

    reg full, empty, almostfull, almostempty, underflow;

    reg [FIFO\_WIDTH-1:0] data\_out\_ref;

    reg wr\_ack\_ref, overflow\_ref;

    reg full\_ref, empty\_ref, almostfull\_ref, almostempty\_ref, underflow\_ref;

    modport DUT (

        input clk,

        input data\_in, rst\_n, wr\_en, rd\_en,

        output data\_out,

        output wr\_ack, overflow,

        output full, empty, almostfull, almostempty, underflow

    );

    modport REF (

        input clk,

        input data\_in, rst\_n, wr\_en, rd\_en,

        output data\_out\_ref,

        output wr\_ack\_ref, overflow\_ref,

        output full\_ref, empty\_ref, almostfull\_ref, almostempty\_ref, underflow\_ref

    );

    modport TEST (

        input clk,

        input data\_out,

        input wr\_ack, overflow,

        input full, empty, almostfull, almostempty, underflow,

        input data\_out\_ref,

        input wr\_ack\_ref, overflow\_ref,

        input full\_ref, empty\_ref, almostfull\_ref, almostempty\_ref, underflow\_ref,

        output data\_in, rst\_n, wr\_en, rd\_en

    );

    modport MONITOR (

        input clk,

        input data\_in, rst\_n, wr\_en, rd\_en,

        input data\_out,

        input wr\_ack, overflow,

        input full, empty, almostfull, almostempty, underflow,

        input data\_out\_ref,

        input wr\_ack\_ref, overflow\_ref,

        input full\_ref, empty\_ref, almostfull\_ref, almostempty\_ref, underflow\_ref

    );

endinterface //FIRO\_interface

/// FIFO design after fixing

import shared\_pkg::\*;

module FIFO(FIFO\_interface.DUT F\_if);

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 F\_if.clk or negedge F\_if.rst\_n) begin

    if (!F\_if.rst\_n) begin

        wr\_ptr <= 0;

        F\_if.wr\_ack <= 0; // FIX

        F\_if.overflow <= 0; // FIX

    end

    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

    else begin

        F\_if.wr\_ack <= 0;

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

            F\_if.overflow <= 1;

        else

            F\_if.overflow <= 0;

    end

end

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

    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

end

always @(posedge F\_if.clk or negedge F\_if.rst\_n) begin

    if (!F\_if.rst\_n) begin

        count <= 0;

    end

    else begin

        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;

    end

end

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

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

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

assign F\_if.almostempty = (count == 1)? 1 : 0;

**/// \*\* Sequence \*\* //**

`ifdef SIM

// Sequences

sequence same\_seq;

    (F\_if.rd\_en && F\_if.wr\_en && !F\_if.empty && !F\_if.full);

endsequence

sequence almE\_wr\_Nrd; // almostempty and write but NOT read

    (F\_if.almostempty && F\_if.wr\_en && !F\_if.rd\_en);

endsequence

sequence F\_wr; // full and write

    (F\_if.full && F\_if.wr\_en);

endsequence

sequence E\_wr; // empty and write

    (F\_if.empty && F\_if.wr\_en);

endsequence

sequence E\_rd; // empty and write

    (F\_if.empty && F\_if.rd\_en);

endsequence

sequence NF\_wr; // NOT full and write

    (!F\_if.full && F\_if.wr\_en);

endsequence

sequence F\_rd; // full and read

    (F\_if.full && F\_if.rd\_en);

endsequence

sequence almE\_Nwr\_rd; // alostempty and read but NOT write

    (F\_if.almostempty && !F\_if.wr\_en && F\_if.rd\_en);

endsequence

sequence almF\_Nwr\_rd; // alostfull and read but NOT write

    (F\_if.almostempty && !F\_if.wr\_en && F\_if.rd\_en);

endsequence

sequence almF\_wr\_Nrd; // almostfull and write but NOT read

    (F\_if.wr\_en && F\_if.almostfull && !F\_if.rd\_en);

endsequence

sequence w\_ptr\_seq;

    (F\_if.wr\_en && count < FIFO\_DEPTH);

endsequence

sequence inc\_rd\_ptr\_rslt;

    ($past(rd\_ptr)+1 == rd\_ptr)||($past(rd\_ptr)==FIFO\_DEPTH-1);

    // ($past(rd\_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(rd\_ptr) = 7 then $past(rd\_ptr) + 1 = 0 (3-bit)

    // for questa $past(rd\_ptr) + 1 = 8

endsequence

**/// \*\* Assertion Internal signal and reset assertion \*\*//**

// ASSERTION

// Internal  signal assertion

count\_inc: assert property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) (F\_if.wr\_en && !F\_if.rd\_en && !F\_if.full) |=> ($past(count)+1 == count));

count\_dec: assert property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) (!F\_if.wr\_en && F\_if.rd\_en && !F\_if.empty)|=> ($past(count)-1 == count));

count\_noChange: assert property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) same\_seq |=> ($past(count) == count));

    inc\_wr\_ptr\_assert:  assert property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n)  w\_ptr\_seq |=> ($past(wr\_ptr)+1 == wr\_ptr) || ($past(wr\_ptr) == FIFO\_DEPTH-1) );

    inc\_rd\_ptr\_assert:  assert property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) (F\_if.rd\_en && count != 0) |=> inc\_rd\_ptr\_rslt );

    always\_comb begin : rst\_n\_assert

        if (!F\_if.rst\_n) begin

            assert\_count\_rst:  assert final (count == 0);

            assert\_wr\_ptr\_rst: assert final (wr\_ptr == 0);

            assert\_rd\_ptr\_rst: assert final (rd\_ptr == 0);

            assert\_wr\_akc\_rst:    assert final (F\_if.wr\_ack == 0);

            assert\_overflow\_rst:  assert final (F\_if.overflow == 0);

            assert\_underflow\_rst: assert final (F\_if.underflow == 0);

            assert\_data\_out\_rst:  assert final (F\_if.data\_out == 0);

            assert\_full\_rst:        assert final (F\_if.full == 0);

            assert\_almostfull\_rst:  assert final (F\_if.almostfull == 0);

            assert\_empty\_rst:       assert final (F\_if.empty == 0);

            assert\_almostempty\_rst: assert final (F\_if.almostempty == 0);

        end

    end

**/// \*\* Global signal Assertion \*\*//**

// Global signal assertion

// full

full\_count:      assert property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) (count >= FIFO\_DEPTH) |->  F\_if.full);

full\_from\_almost:assert property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) almF\_wr\_Nrd |=> F\_if.full);

full\_noChange:   assert property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) same\_seq    |=> !F\_if.full);

full\_inactive:   assert property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) F\_rd        |=> !F\_if.full);

// wr\_ack

ack\_active:   assert property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) NF\_wr |=> F\_if.wr\_ack);

ack\_inactive: assert property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) F\_wr  |=> !F\_if.wr\_ack);

// almostfull

almostfull\_count:    assert property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) (count==FIFO\_DEPTH-1) |-> F\_if.almostfull);

almostfull\_from\_full:assert property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) F\_rd  |=> ($fell(F\_if.full) && $rose(F\_if.almostfull)));

almostfull\_noChange: assert property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) same\_seq |=> $past(F\_if.almostfull) == F\_if.almostfull);

almostfull\_inactive:  assert property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) almF\_Nwr\_rd |=> !F\_if.almostfull );

// overflow

overflow\_active:assert property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) F\_wr |=> F\_if.overflow);

overflow\_wr\_in: assert property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) ($past(F\_if.overflow) && !F\_if.wr\_en) |=> !F\_if.overflow);

overflow\_Nfull:assert property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) ($past(F\_if.overflow) && !F\_if.full)  |=> !F\_if.overflow);

// almostempty

almostempty\_count:     assert property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) (count==1) |-> F\_if.almostempty );

almostempty\_noChange:  assert property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) same\_seq |=> ($past(F\_if.almostempty) == F\_if.almostempty));

almostempty\_from\_empty:assert property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) E\_wr |=> ($fell(F\_if.empty) && $rose(F\_if.almostempty)));

almostempty\_inactive:  assert property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) almE\_wr\_Nrd |=> !F\_if.almostempty );

// empty

empty\_count:      assert property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) (count==ZERO) |-> F\_if.empty );

empty\_noChaneg:   assert property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) same\_seq |=> ($past(F\_if.empty) == F\_if.empty) );

empty\_from\_almost:assert property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) almE\_Nwr\_rd |=> F\_if.empty);

empty\_inactive:   assert property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) E\_wr |=> !F\_if.empty);

// underflow

underflow\_active:  assert property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) E\_rd |=> F\_if.underflow);

underflow\_Nempty:assert property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) ($past(F\_if.underflow) && !F\_if.empty)  |=> !F\_if.underflow);

underflow\_Nrd:   assert property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) ($past(F\_if.underflow) && !F\_if.rd\_en)  |=> !F\_if.underflow);

`endif

endmodule

\

**Coverage Directives**

// Coverage

// Internal  signal assertion

count\_inc\_cover: cover property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) (F\_if.wr\_en && !F\_if.rd\_en && !F\_if.full) |=> ($past(count)+1 == count));

count\_dec\_cover: cover property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) (!F\_if.wr\_en && F\_if.rd\_en && !F\_if.empty)|=> ($past(count)-1 == count));

count\_noChange\_cover: cover property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) same\_seq |=> ($past(count) == count));

inc\_wr\_ptr\_cover:  cover property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n)  w\_ptr\_seq |=> ($past(wr\_ptr)+1 == wr\_ptr) || ($past(wr\_ptr) == FIFO\_DEPTH-1) );

inc\_rd\_ptr\_cover:  cover property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) (F\_if.rd\_en && count != 0) |=> inc\_rd\_ptr\_rslt );

    always\_comb begin : rst\_n\_cover

        if (!F\_if.rst\_n) begin

            cover\_count\_rst:  cover final (count == 0);

            cover\_wr\_ptr\_rst: cover final (wr\_ptr == 0);

            cover\_rd\_ptr\_rst: cover final (rd\_ptr == 0);

            cover\_wr\_akc\_rst:    cover final (F\_if.wr\_ack == 0);

            cover\_overflow\_rst:  cover final (F\_if.overflow == 0);

            cover\_underflow\_rst: cover final (F\_if.underflow == 0);

            cover\_data\_out\_rst:  cover final (F\_if.data\_out == 0);

            cover\_full\_rst:        cover final (F\_if.full == 0);

            cover\_almostfull\_rst:  cover final (F\_if.almostfull == 0);

            cover\_empty\_rst:       cover final (F\_if.empty == 0);

            cover\_almostempty\_rst: cover final (F\_if.almostempty == 0);

        end

    end

// Global signal cover

// full

full\_count\_cover:    cover property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) (count >= FIFO\_DEPTH) |->  F\_if.full);

full\_from\_almost\_cover:cover property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) almF\_wr\_Nrd |=> F\_if.full);

full\_noChange\_cover:     cover property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) same\_seq    |=> !F\_if.full);

full\_inactive\_cover:     cover property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) F\_rd     |=> !F\_if.full);

// wr\_ack

ack\_active\_cover:   cover property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) NF\_wr |=> F\_if.wr\_ack);

ack\_inactive\_cover: cover property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) F\_wr  |=> !F\_if.wr\_ack);

// almostfull

almostfull\_count\_cover:  cover property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) (count==FIFO\_DEPTH-1) |-> F\_if.almostfull);

almostfull\_from\_full\_cover:cover property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) F\_rd  |=> ($fell(F\_if.full) && $rose(F\_if.almostfull)));

almostfull\_noChange\_cover: cover property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) same\_seq |=> $past(F\_if.almostfull) == F\_if.almostfull);

almostfull\_inactive\_cover:  cover property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) almF\_Nwr\_rd |=> !F\_if.almostfull );

// overflow

overflow\_active\_cover:cover property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) F\_wr |=> F\_if.overflow);

overflow\_wr\_in\_cover: cover property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) ($past(F\_if.overflow) && !F\_if.wr\_en) |=> !F\_if.overflow);

overflow\_Nfull\_cover:cover property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) ($past(F\_if.overflow) && !F\_if.full)  |=> !F\_if.overflow);

// almostempty

almostempty\_count\_cover:       cover property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) (count==1) |-> F\_if.almostempty );

almostempty\_noChange\_cover:  cover property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) same\_seq |=> ($past(F\_if.almostempty) == F\_if.almostempty));

almostempty\_from\_empty\_cover:cover property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) E\_wr |=> ($fell(F\_if.empty) && $rose(F\_if.almostempty)));

almostempty\_inactive\_cover:  cover property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) almE\_wr\_Nrd |=> !F\_if.almostempty );

// empty

empty\_count\_cover:    cover property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) (count==ZERO) |-> F\_if.empty );

empty\_noChaneg\_cover:   cover property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) same\_seq |=> ($past(F\_if.empty) == F\_if.empty) );

empty\_from\_almost\_cover:cover property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) almE\_Nwr\_rd |=> F\_if.empty);

empty\_inactive\_cover:   cover property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) E\_wr |=> !F\_if.empty);

// underflow

underflow\_active\_cover:  cover property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) E\_rd |=> F\_if.underflow);

underflow\_Nempty\_cover:cover property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) ($past(F\_if.underflow) && !F\_if.empty)  |=> !F\_if.underflow);

underflow\_Nrd\_cover:   cover property (@(posedge F\_if.clk) disable iff(!F\_if.rst\_n) ($past(F\_if.underflow) && !F\_if.rd\_en)  |=> !F\_if.underflow);

`endif

endmodule

**/// shared\_pkg**

package shared\_pkg;

        parameter FIFO\_WIDTH = 16;

        parameter FIFO\_DEPTH = 8;

        parameter ACTIVE = 1;

        parameter INACTIVE = 0;

        int WR\_EN\_ON\_DIST = 70;

        int RD\_EN\_ON\_DIST = 30;

        int RESET\_ACTIVE = 5;

        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};

        //bit [FIFO\_WIDTH-1:0] one\_bit\_high [16] = '{16'b0000\_0000\_0000\_0001,

        //                                           16'b0000\_0000\_0000\_0010,

        //                                           16'b0000\_0000\_0000\_0100,

        //                                           16'b0000\_0000\_0000\_1000,

        //                                           16'b0000\_0000\_0001\_0000,

        //                                           16'b0000\_0000\_0010\_0000,

        //                                           16'b0000\_0000\_0100\_0000,

        //                                           16'b0000\_0000\_1000\_0000,

        //                                           16'b0000\_0001\_0000\_0000,

        //                                           16'b0000\_0010\_0000\_0000,

        //                                           16'b0000\_0100\_0000\_0000,

        //                                           16'b0000\_1000\_0000\_0000,

        //                                           16'b0001\_0000\_0000\_0000,

        //                                           16'b0010\_0000\_0000\_0000,

        //                                           16'b0100\_0000\_0000\_0000,

        //                                           16'b1000\_0000\_0000\_0000}

        bit test\_finished;

        int error\_counter = 0;

        int correct\_counter = 0;

        int inc\_correct\_counter = 0;

        int LOOP0 = 1\_00;

        int LOOP1 = 1\_000;

        int LOOP2 = 10\_000;

        int LOOP3 = 100\_000;

endpackage

**/// transaction\_pkg**

package transaction\_pkg;

import shared\_pkg::\*;

class FIFO\_transaction;

// input

    rand bit [FIFO\_WIDTH-1:0] data\_in;

    rand bit rst\_n, wr\_en, rd\_en;

    // bit clk;

// output

    bit [FIFO\_WIDTH-1:0] data\_out;

    bit wr\_ack, overflow;

    bit full, empty, almostfull, almostempty, underflow;

bit toggle;

// Costraint Block

    // requirement constraint

    constraint CON\_RESET {

        rst\_n dist {1:=100-RESET\_ACTIVE, 0:=RESET\_ACTIVE};

    }

    constraint CON\_W\_R\_DIST {

        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};

    }

    // constraint to assert onley write with constraint one bit high to data\_in

    constraint MY\_CON\_ONLEY\_W {

        wr\_en == ACTIVE;

        rd\_en == INACTIVE;

    }

    // constraint to assert onley read with constraint zero or MAX ro data\_in

    constraint MY\_CON\_ONLEY\_R {

        wr\_en == INACTIVE;

        rd\_en == ACTIVE;

    }

    // constraint to Toggle write and read enable

    constraint MY\_CON\_OPPSITE {

        wr\_en == ~rd\_en;

    }

    // constraint to make write and read active at the same time

    constraint MY\_CON\_BOTH\_ACTIVE {

        wr\_en == ACTIVE;

        rd\_en == ACTIVE;

    }

    constraint CON\_DATA\_OUT\_ONE\_BIT{

$countones(data\_in) == 1;

    }

    constraint CON\_DATA\_OUT\_M\_Z{

        ((data\_in == ZERO) || (data\_in == MAX));

    }

// Mythods

    int wr\_old = 0, rd\_old = 0;

    function void pre\_randomize();

        wr\_old = wr\_en;

        rd\_old = rd\_en;

    endfunction

    // I want to assert rst for the first 5 clk cycle

    int first\_rst = 0;

    function void post\_randomize();

        if (first\_rst <= 5) begin

            first\_rst++;

            rst\_n = 0;

        end

        if (toggle) begin

            wr\_en = ~wr\_old;

            rd\_en = ~rd\_old;

        end

    endfunction

    function void assigned\_values(

        //  output

        output bit [FIFO\_WIDTH-1:0] data\_in,

        output bit rst\_n, wr\_en, rd\_en,

        // input

        input bit [FIFO\_WIDTH-1:0] data\_out,

        input bit wr\_ack, overflow,

        input bit full, empty, almostfull, almostempty, underflow

    );

        // output

        data\_in = this.data\_in;

        rst\_n   = this.rst\_n;

        wr\_en   = this.wr\_en;

        rd\_en   = this.rd\_en;

        // input

        this.data\_out    = data\_out;

        this.wr\_ack      = wr\_ack;

        this.overflow    = overflow;

        this.full        = full;

        this.empty       = empty;

        this.almostfull  = almostfull;

        this.almostempty = almostempty;

        this.underflow   = underflow;

    endfunction

    function new();

    endfunction //new()

endclass //FIFO\_transaction

endpackage

/// scoreboard\_pkg

package scoreboard\_pkg;

import transaction\_pkg::\*;

import shared\_pkg::\*;

class FIFO\_scoreboard;

// input

    bit [FIFO\_WIDTH-1:0] data\_in;

    bit rst\_n, wr\_en, rd\_en;

// output refrence

    bit [FIFO\_WIDTH-1:0] data\_out\_ref;

    bit wr\_ack\_ref, overflow\_ref;

    bit full\_ref, empty\_ref, almostfull\_ref;

    bit almostempty\_ref, underflow\_ref;

    function void reference\_model(

        input bit [FIFO\_WIDTH-1:0] data\_out\_,

        input bit wr\_ack\_, overflow\_, full\_, empty\_,

        input bit almostfull\_, almostempty\_, underflow\_

    );

        data\_out\_ref     = data\_out\_;

        wr\_ack\_ref       = wr\_ack\_;

        overflow\_ref     = overflow\_;

        full\_ref         = full\_;

        empty\_ref        = empty\_;

        almostfull\_ref   = almostfull\_;

        almostempty\_ref  = almostempty\_;

        underflow\_ref    = underflow\_;

    endfunction // reference\_model()

    function void check\_data (input FIFO\_transaction F\_txn);

        if (data\_out\_ref != F\_txn.data\_out) begin

            $error("%t: Error - data\_out\_ref = %0d, data\_out = %0d",$time, data\_out\_ref, F\_txn.data\_out);

            inc\_correct\_counter++;

        end

        display\_errors(wr\_ack\_ref, F\_txn.wr\_ack, "wr\_ack");

        display\_errors(full\_ref, F\_txn.full, "full");

        display\_errors(empty\_ref, F\_txn.empty, "empty");

        display\_errors(almostfull\_ref, F\_txn.almostfull, "almostfull");

        display\_errors(almostempty\_ref, F\_txn.almostempty, "almostempty");

        display\_errors(overflow\_ref, F\_txn.overflow, "overflow");

        display\_errors(underflow\_ref, F\_txn.underflow, "underflow");

        if (inc\_correct\_counter == 7) begin

            correct\_counter++;

            inc\_correct\_counter = 0;

        end

    endfunction //check\_data ()

    function void display\_errors (input bit gld, dut, input string x);

        if (gld!=dut) begin

            $error("%s = %0d, %s\_ref = %0d", x, dut, x, gld);

            error\_counter++;

        end else

            inc\_correct\_counter++;

    endfunction

    function new();

    endfunction //new()

endclass //FIFO\_scoreboard

endpackage

/// coverage\_pkg

package coverage\_pkg;

import transaction\_pkg::\*;

import shared\_pkg::\*;

class FIFO\_coverage;

FIFO\_transaction F\_cvg\_txn = new();

covergroup CVG;

// rst\_n coverage

    rst\_cp: coverpoint F\_cvg\_txn.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 F\_cvg\_txn.wr\_en{

        bins active = {1};

        bins inactive = {0};

    }

    rd\_en\_cp:        coverpoint F\_cvg\_txn.rd\_en{

        bins active = {1};

        bins inactive = {0};

    }

// data\_out bus coverpoint

    data\_out\_cp:     coverpoint F\_cvg\_txn.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 F\_cvg\_txn.wr\_ack{

        bins active = {1};

        bins inactive = {0};

        bins inactive\_to\_active = (0 => 1);

        bins active\_to\_inactive = (1 => 0);

    }

    full\_cp:         coverpoint F\_cvg\_txn.full{

        bins active = {1};

        bins inactive = {0};

        bins active\_to\_inactive = (1 => 0);

        bins inactive\_to\_active = (0 => 1);

    }

    empty\_cp:        coverpoint F\_cvg\_txn.empty{

        bins active = {1};

        bins inactive = {0};

        bins active\_to\_inactive = (1 => 0);

        bins inactive\_to\_active = (0 => 1);

    }

    almostfull\_cp:   coverpoint F\_cvg\_txn.almostfull{

        bins active = {1};

        bins inactive = {0};

        bins active\_to\_inactive = (1 => 0);

        bins inactive\_to\_active = (0 => 1);

    }

    almostempty\_cp:  coverpoint F\_cvg\_txn.almostempty{

        bins active = {1};

        bins inactive = {0};

        bins active\_to\_inactive = (1 => 0);

        bins inactive\_to\_active = (0 => 1);

    }

    underflow\_cp:    coverpoint F\_cvg\_txn.underflow{

        bins active = {1};

        bins inactive = {0};

    }

    overflow\_cp:     coverpoint F\_cvg\_txn.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\_ack = 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;

    // 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 A\_ack\_A\_wr\_I\_full = binsof(wr\_ack\_cp.active)

                                && binsof(wr\_en\_cp.active)

                                && binsof(full\_cp.inactive);

        bins ack\_full = binsof(wr\_ack\_cp.active) && binsof(full\_cp.inactive);

        bins A\_ack\_A\_full\_trans = 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 emptyTRANS\_ackAC = 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 Aack\_Ialmostempty = 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 trans\_rst\_full = 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;

    // 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 = 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 {

        bins rst\_empty = binsof(empty\_cp.active) && binsof(rst\_cp.active);

        option.cross\_auto\_bin\_max = 0;

    }

    // 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;

    // 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\_ack = 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, rd\_en\_cp, overflow\_cp{

        bins both\_high = binsof(wr\_en\_cp.active) && binsof(overflow\_cp.active);

    }

// 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 wr\_en\_cp, rd\_en\_cp, underflow\_cp{

        bins both\_high = binsof(rd\_en\_cp.active) && binsof(underflow\_cp.active);

    }

// 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 both\_Wr\_high = binsof(wr\_en\_cp.active) && binsof(almostempty\_cp.active);

        bins INalmostEmp\_Awr = binsof(wr\_en\_cp.active) && binsof(almostempty\_cp.inactive);

        bins both\_Re\_high = binsof(rd\_en\_cp.active) && binsof(almostempty\_cp.active);

        bins INalmostEmp\_Ard = binsof(rd\_en\_cp.active) && binsof(almostempty\_cp.inactive);

    }

// 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 both\_Wr\_high = binsof(wr\_en\_cp.active) && binsof(almostfull\_cp.active);

        bins INalmostfull\_Awr = binsof(wr\_en\_cp.active) && binsof(almostfull\_cp.inactive);

        bins both\_Re\_high = binsof(rd\_en\_cp.active) && binsof(almostfull\_cp.active);

        bins INalmostfull\_Ard = binsof(rd\_en\_cp.active) && binsof(almostfull\_cp.inactive);

    }

// 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

// methods

    function new();

        CVG = new();

    endfunction //new()

    function void sample\_data(FIFO\_transaction F\_txn);

        F\_cvg\_txn = F\_txn; // shallow copy

        CVG.sample();

    endfunction //sample\_data()

endclass //FIFO\_coverage

endpackage

/// golden model design

import shared\_pkg::\*;

module FIFO\_ref(FIFO\_interface.REF F\_if);

    reg [FIFO\_WIDTH-1:0] fifoo\_q [$];

    // Write

    always @(posedge F\_if.clk or negedge F\_if.rst\_n) begin

        if (!F\_if.rst\_n) begin

            fifoo\_q.delete();

            F\_if.wr\_ack\_ref <= 0;

            F\_if.overflow\_ref <= 0;

        end

        else if (F\_if.wr\_en && !F\_if.full\_ref) begin

            fifoo\_q.push\_back(F\_if.data\_in);

            F\_if.wr\_ack\_ref <= 1;

            F\_if.overflow\_ref <= 0;

        end

        else begin

            F\_if.wr\_ack\_ref <= 0;

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

                F\_if.overflow\_ref <= 1;

            else

                F\_if.overflow\_ref <= 0;

        end

    end

    // Read

    always @(posedge F\_if.clk or negedge F\_if.rst\_n) begin

        if (!F\_if.rst\_n) begin

            fifoo\_q.delete();

            F\_if.underflow\_ref <= 0;

            F\_if.data\_out\_ref <= 0;

        end

        else if (F\_if.rd\_en && !F\_if.empty\_ref) begin

            F\_if.data\_out\_ref <= fifoo\_q.pop\_front();

            F\_if.underflow\_ref <= 0;

        end

        else begin

            if (F\_if.empty\_ref && F\_if.rd\_en)

                F\_if.underflow\_ref <= 1;

            else

                F\_if.underflow\_ref <= 0;

        end

    end

    assign F\_if.almostfull\_ref = (fifoo\_q.size() == FIFO\_DEPTH-1)? 1:0;

    assign F\_if.full\_ref = (fifoo\_q.size() >= FIFO\_DEPTH)? 1:0;

    assign F\_if.empty\_ref = (fifoo\_q.size() == 0 && F\_if.rst\_n)? 1:0;

    assign F\_if.almostempty\_ref = (fifoo\_q.size() == 1)? 1:0;

endmodule

/// monitor

import shared\_pkg::\*;

import scoreboard\_pkg::\*;

import transaction\_pkg::\*;

import coverage\_pkg::\*;

module monitor (FIFO\_interface.MONITOR F\_if);

    FIFO\_transaction tr;

    FIFO\_scoreboard sb;

    FIFO\_coverage cov;

initial begin

    tr = new();

    sb = new();

    cov = new();

    forever begin

        @(negedge F\_if.clk) begin

            // input

            tr.data\_in = F\_if.data\_in;

            tr.rst\_n   = F\_if.rst\_n;

            tr.wr\_en   = F\_if.wr\_en;

            tr.rd\_en   = F\_if.rd\_en;

            // output

            tr.data\_out    = F\_if.data\_out;

            tr.wr\_ack      = F\_if.wr\_ack;

            tr.overflow    = F\_if.overflow;

            tr.full        = F\_if.full;

            tr.empty       = F\_if.empty;

            tr.almostfull  = F\_if.almostfull;

            tr.almostempty = F\_if.almostempty;

            tr.underflow   = F\_if.underflow;

        end

        fork

            // Process 1

            begin

                sb.reference\_model(

                    F\_if.data\_out\_ref, F\_if.wr\_ack\_ref, F\_if.overflow\_ref,

                    F\_if.full\_ref, F\_if.empty\_ref, F\_if.almostfull\_ref,

                    F\_if.almostempty\_ref, F\_if.underflow\_ref

                );

                sb.check\_data(tr);

            end

            // Process 2

            begin

                cov.sample\_data(tr);

            end

        join

        if (test\_finished) begin

            $display("");

            $display("\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*");

            $display("\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\* summary: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*");

            $display("\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*");

            $display("\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*");

            $display("\*\*\*\*\*\*\*\*\*\*\* error\_counter = %0d, correct\_counter = %0d \*\*\*\*\*\*\*\*\*\*\*",error\_counter, correct\_counter);

            $stop;

        end

    end

end

endmodule

/// testbench

import shared\_pkg::\*;

import transaction\_pkg::\*;

module testbench (FIFO\_interface.TEST F\_if);

FIFO\_transaction tr = new();

initial begin

// All constraint

// tr.CONSTRAINT\_NAME.constraint\_mode(0); => Turn OFF specific constraint

// tr.CONSTRAINT\_NAME.constraint\_mode(1); => Turn ON specific constraint

// disable all constraint

tr.constraint\_mode(0);

RESET\_ACTIVE = 10;

// turn on reset constraint in all tb cases

tr.CON\_RESET.constraint\_mode(1);

// turn ON requirement constraint

tr.CON\_W\_R\_DIST.constraint\_mode(1);

$display("%0t: Turn ON CON\_W\_R\_DIST",$time);

tr.CON\_DATA\_OUT\_ONE\_BIT.constraint\_mode(1); // ON constraint data\_in to be only one bit high

$display("%0t: Turn ON CON\_DATA\_OUT\_ONE\_BIT",$time);

    repeat(LOOP2) randomization;

$display("%0t: Turn OFF CON\_DATA\_OUT\_ONE\_BIT",$time);

tr.CON\_DATA\_OUT\_ONE\_BIT.constraint\_mode(0); // OFF constraint data\_in to be only one bit high

tr.CON\_DATA\_OUT\_M\_Z.constraint\_mode(1); // ON constraint data\_in to be MAX or ZERO

$display("%0t: Turn ON CON\_DATA\_OUT\_M\_Z ",$time);

    repeat(LOOP1) randomization;

$display("%0t: Turn OFF CON\_DATA\_OUT\_M\_Z ",$time);

tr.CON\_DATA\_OUT\_M\_Z.constraint\_mode(0); // OFF constraint data\_in to be MAX or ZERO

$display("%0t: Turn OFF CON\_W\_R\_DIST",$time);

tr.CON\_W\_R\_DIST.constraint\_mode(0); // turn OFF requirement constraint

// turn on MY\_CON\_ONLEY\_W constraint => wr\_en always active, rd\_en always inactive

tr.MY\_CON\_ONLEY\_W.constraint\_mode(1);

$display("%0t: Turn ON MY\_CON\_ONLEY\_W",$time);

    repeat(FIFO\_DEPTH) randomization;

$display("%0t: Turn OFF MY\_CON\_ONLEY\_W",$time);

tr.MY\_CON\_ONLEY\_W.constraint\_mode(0);

// reset

reset;

// turn on MY\_CON\_ONLEY\_W constraint => wr\_en always active, rd\_en always inactive

tr.MY\_CON\_ONLEY\_W.constraint\_mode(1);

$display("%0t: Turn ON MY\_CON\_ONLEY\_W",$time);

    repeat(FIFO\_DEPTH \* 4) randomization;

$display("%0t: Turn OFF MY\_CON\_ONLEY\_W",$time);

tr.MY\_CON\_ONLEY\_W.constraint\_mode(0);

// turn on MY\_CON\_ONLEY\_R constraint => rd\_en always active, wr\_en always inactive

tr.MY\_CON\_ONLEY\_R.constraint\_mode(1);

$display("%0t: Turn ON MY\_CON\_ONLEY\_R",$time);

    repeat(FIFO\_DEPTH-1) randomization;

$display("%0t: Turn OFF MY\_CON\_ONLEY\_R",$time);

tr.MY\_CON\_ONLEY\_R.constraint\_mode(0);

// turn on MY\_CON\_BOTH\_ACTIVE constraint => rd\_en always active, wr\_en always active

tr.MY\_CON\_BOTH\_ACTIVE.constraint\_mode(1);

$display("%0t: Turn ON MY\_CON\_BOTH\_ACTIVE",$time);

    repeat(LOOP1) randomization;

$display("%0t: Turn OFF MY\_CON\_BOTH\_ACTIVE",$time);

tr.MY\_CON\_BOTH\_ACTIVE.constraint\_mode(0);

// turn on MY\_CON\_OPPSITE constraint => rd\_en always invert of wr\_en

tr.MY\_CON\_OPPSITE.constraint\_mode(1);

$display("%0t: Turn ON MY\_CON\_OPPSITE",$time);

    repeat(LOOP1) randomization;

$display("%0t: Turn OFF MY\_CON\_OPPSITE",$time);

tr.MY\_CON\_OPPSITE.constraint\_mode(0);

// TURN on toggle => wr\_en and rd\_en toggle every clk cycle

tr.toggle = 1;

$display("%0t: Turn ON Toggle",$time);

repeat(LOOP1) randomization;

$display("%0t: Turn OFF Toggle",$time);

tr.toggle = 0;

// turn on MY\_CON\_ONLEY\_R constraint => rd\_en always active, wr\_en always inactive

tr.MY\_CON\_ONLEY\_R.constraint\_mode(1);

$display("%0t: Turn ON MY\_CON\_ONLEY\_R",$time);

    repeat(FIFO\_DEPTH \* 5) randomization;

$display("%0t: Turn OFF MY\_CON\_ONLEY\_R",$time);

tr.MY\_CON\_ONLEY\_R.constraint\_mode(0);

// turn on MY\_CON\_ONLEY\_W constraint => wr\_en always active, rd\_en always inactive

tr.MY\_CON\_ONLEY\_W.constraint\_mode(1);

$display("%0t: Turn ON MY\_CON\_ONLEY\_W",$time);

    repeat(FIFO\_DEPTH - 1) randomization;

$display("%0t: Turn OFF MY\_CON\_ONLEY\_W",$time);

tr.MY\_CON\_ONLEY\_W.constraint\_mode(0);

// turn on MY\_CON\_BOTH\_ACTIVE constraint => rd\_en always active, wr\_en always active

tr.MY\_CON\_BOTH\_ACTIVE.constraint\_mode(1);

$display("%0t: Turn ON MY\_CON\_BOTH\_ACTIVE",$time);

    repeat(LOOP1) randomization;

$display("%0t: Turn OFF MY\_CON\_BOTH\_ACTIVE",$time);

tr.MY\_CON\_BOTH\_ACTIVE.constraint\_mode(0);

// turn on MY\_CON\_ONLEY\_W constraint => wr\_en always active, rd\_en always inactive

tr.MY\_CON\_ONLEY\_W.constraint\_mode(1);

$display("%0t: Turn ON MY\_CON\_ONLEY\_W",$time);

    repeat(FIFO\_DEPTH) randomization;

$display("%0t: Turn OFF MY\_CON\_ONLEY\_W",$time);

tr.MY\_CON\_ONLEY\_W.constraint\_mode(0);

// turn on MY\_CON\_ONLEY\_R constraint => rd\_en always active, wr\_en always inactive

tr.MY\_CON\_ONLEY\_R.constraint\_mode(1);

$display("%0t: Turn ON MY\_CON\_ONLEY\_R",$time);

    repeat(FIFO\_DEPTH - 1) randomization;

$display("%0t: Turn OFF MY\_CON\_ONLEY\_R",$time);

tr.MY\_CON\_ONLEY\_R.constraint\_mode(0);

// turn on MY\_CON\_BOTH\_ACTIVE constraint => rd\_en always active, wr\_en always active

tr.MY\_CON\_BOTH\_ACTIVE.constraint\_mode(1);

$display("%0t: Turn ON MY\_CON\_BOTH\_ACTIVE",$time);

    repeat(LOOP0) randomization;

$display("%0t: Turn OFF MY\_CON\_BOTH\_ACTIVE",$time);

tr.MY\_CON\_BOTH\_ACTIVE.constraint\_mode(0);

// reset

reset;

tr.constraint\_mode(0);

WR\_EN\_ON\_DIST = 30; // change probablty of wr\_en get high

RD\_EN\_ON\_DIST = 70; // change probablty of rd\_en get high

tr.CON\_W\_R\_DIST.constraint\_mode(1);

$display("%0t: Turn ON CON\_W\_R\_DIST",$time);

    repeat(LOOP2) randomization;

$display("%0t: Turn OFF CON\_W\_R\_DIST",$time);

tr.CON\_W\_R\_DIST.constraint\_mode(0);

tr.constraint\_mode(0);

// reset

reset;

RESET\_ACTIVE = 10; // change probablty of rst\_n get active

// turn of rst constraint

tr.CON\_RESET.constraint\_mode(1);

repeat(LOOP3) randomization;

// Test Finished

test\_finished = 1;

@(negedge F\_if.clk);

//$stop;

end

task randomization;

        assert(tr.randomize());

        assigned\_tr\_itf;

        @(negedge F\_if.clk);

endtask //

task assigned\_tr\_itf;

    F\_if.data\_in = tr.data\_in;

    F\_if.rst\_n   = tr.rst\_n;

    F\_if.wr\_en   = tr.wr\_en;

    F\_if.rd\_en   = tr.rd\_en;

endtask //

task reset;

    F\_if.rst\_n = 0;

    tr.rst\_n = F\_if.rst\_n;

    repeat(3) @(negedge F\_if.clk);

    F\_if.rst\_n = 1;

    tr.rst\_n = F\_if.rst\_n;

endtask //

endmodule

/// top

module top ();

    bit clk;

    initial begin

        forever #1 clk = ~clk;

    end

    FIFO\_interface F\_if(clk);

    FIFO dut(F\_if);

    FIFO\_ref gld(F\_if);

    testbench tb(F\_if);

    monitor mon(F\_if);

endmodule

**/// small testbench to test golden model**

import shared\_pkg::\*;

// Testbench for refrence

module refrence\_tb ();

    logic [FIFO\_WIDTH-1:0] data\_in;

    logic clk, rst\_n, wr\_en, rd\_en;

    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;

    initial begin

        clk = 0;

        forever begin

            #1 clk = ~clk;

        end

    end

    FIFO\_ref dut(.\*);

    int i;

    initial begin

        $readmemh ("fifoo.dat", dut.mem);

        rst\_n = 0; #2; rst\_n = 1; #1;

        rst\_n = 0; #2; rst\_n = 1; #1;

        wr\_en = 1; rd\_en = 0;

        for (i = 0; i<FIFO\_DEPTH; i++) begin

            data\_in = i+1;

            @(posedge clk);

        end

        data\_in = 0;

        repeat(3) @(posedge clk);

        wr\_en = 0; rd\_en = 1;

        repeat(FIFO\_DEPTH) begin

            @(posedge clk);

        end

        repeat(3) @(posedge clk);

        wr\_en = 1; rd\_en = 0;

        for (i = 0; i<5; i++) begin

            data\_in = i+1;

            @(posedge clk);

        end

        wr\_en = 1; rd\_en = 1;

        for (i = 5; i<25; i++) begin

            data\_in = 25-i;

            @(posedge clk);

        end

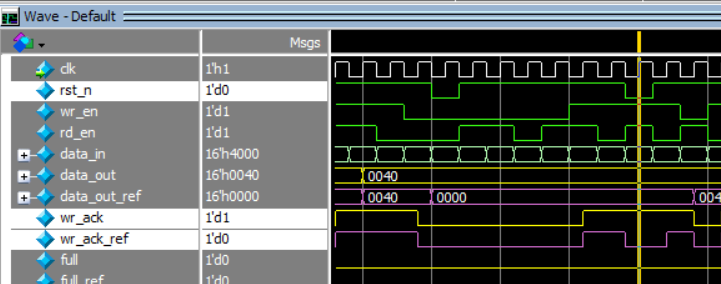
    $stop;

    end

endmodule

**/// \*\* 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

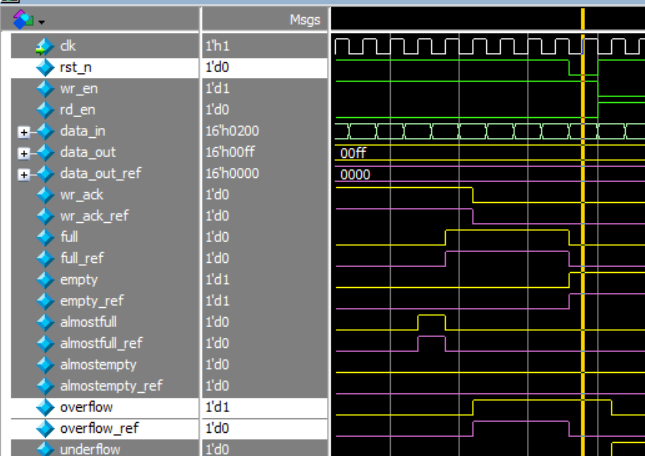
        wr\_ptr <= 0;

        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;

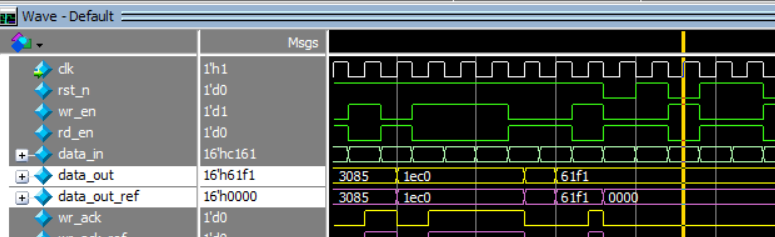
        F\_if.wr\_ack <= 0; //

        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

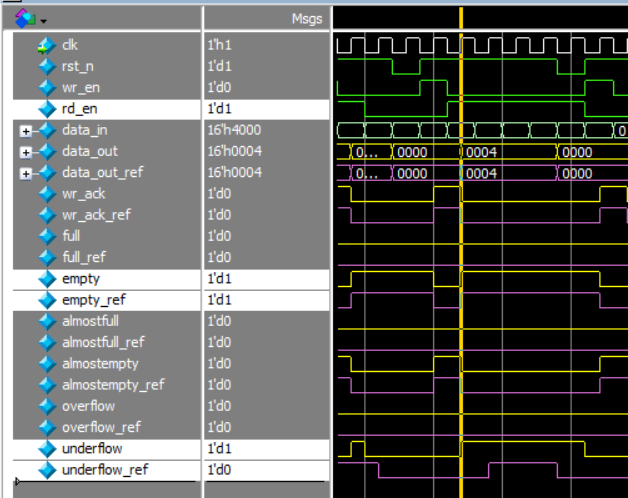
**/// \*\* Bug\_4: underflow must be sequential not combinational \*\* ///**

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)

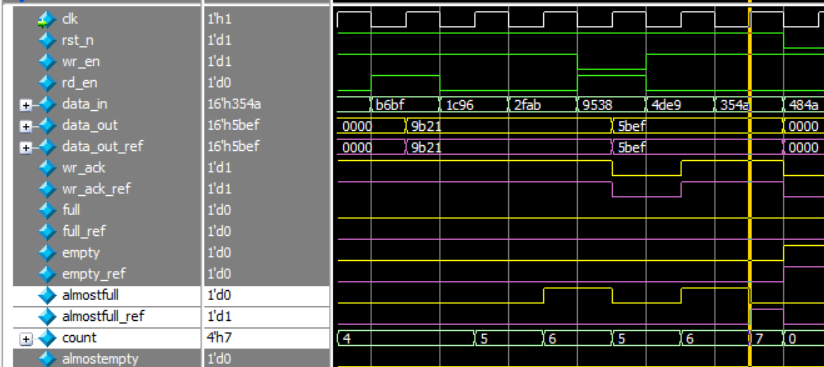
            count <= count + 1;

        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:

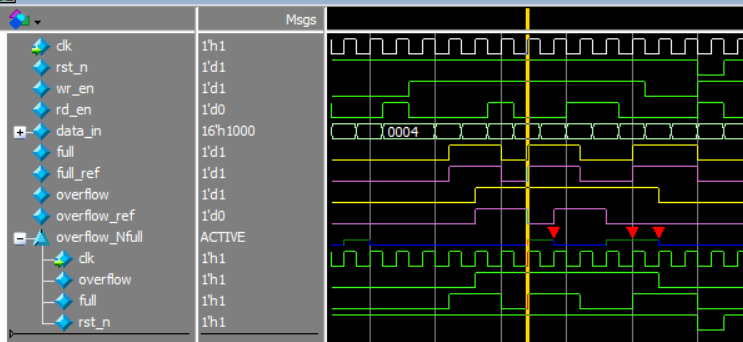
assign F\_if.almostfull = (count == FIFO\_DEPTH-1)? 1 : 0; //

**/// \*\* 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;

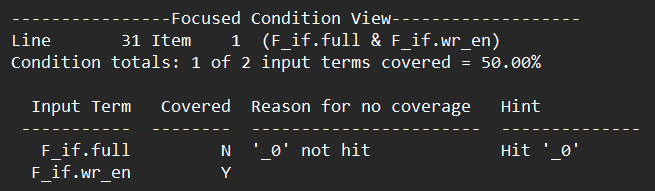
        F\_if.wr\_ack <= 1;

        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\*\* ///**

****

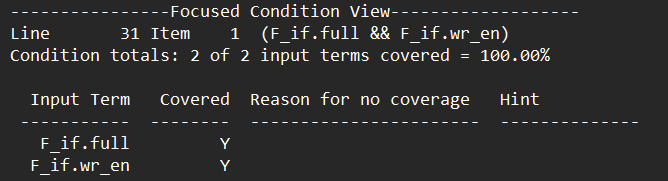
        if (F\_if.full & F\_if.wr\_en)

            F\_if.overflow <= 1;

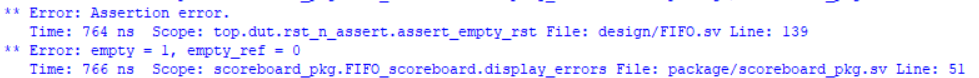
Fix:

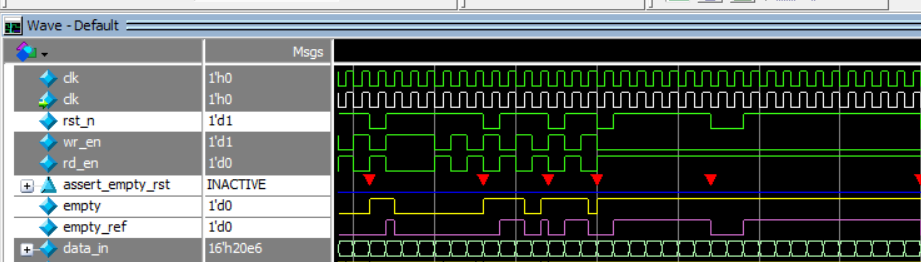
        if (F\_if.full && F\_if.wr\_en)// FIX

            F\_if.overflow <= 1;



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

****

****

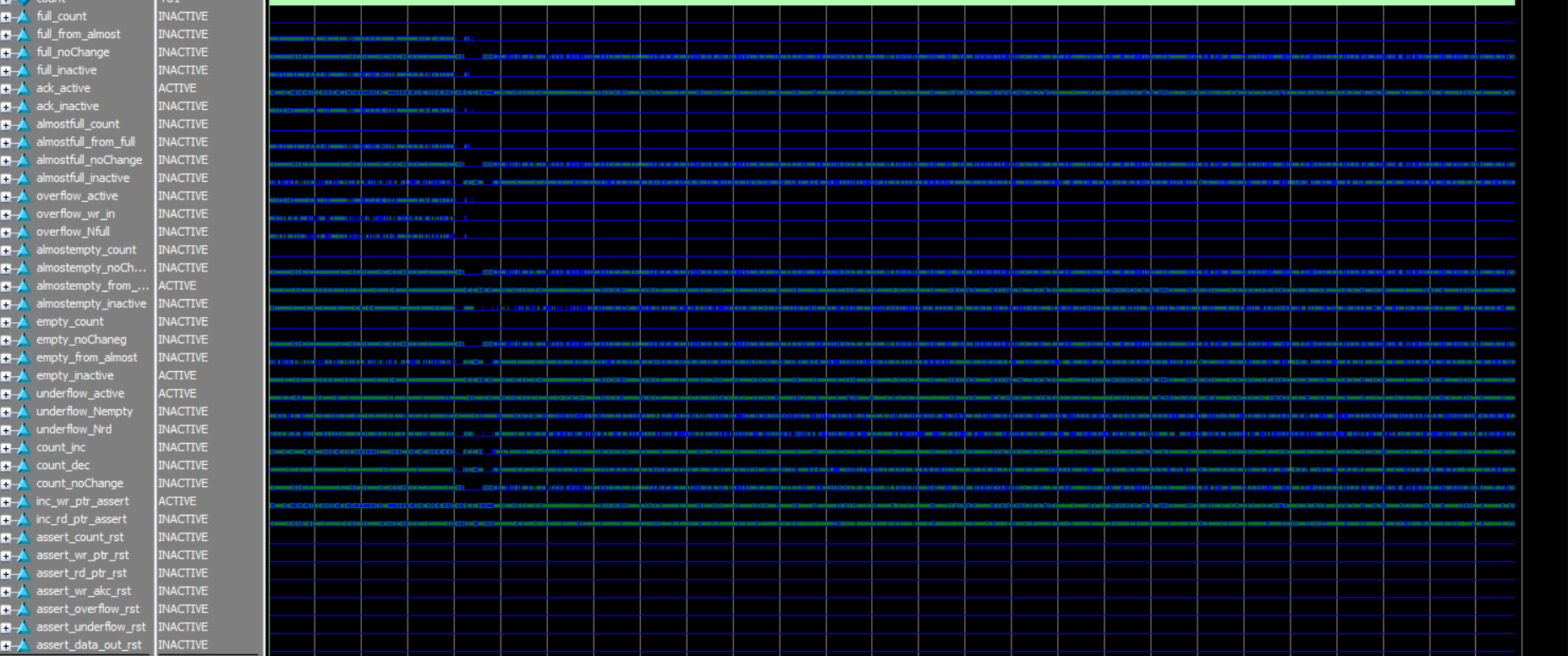
Fix:

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

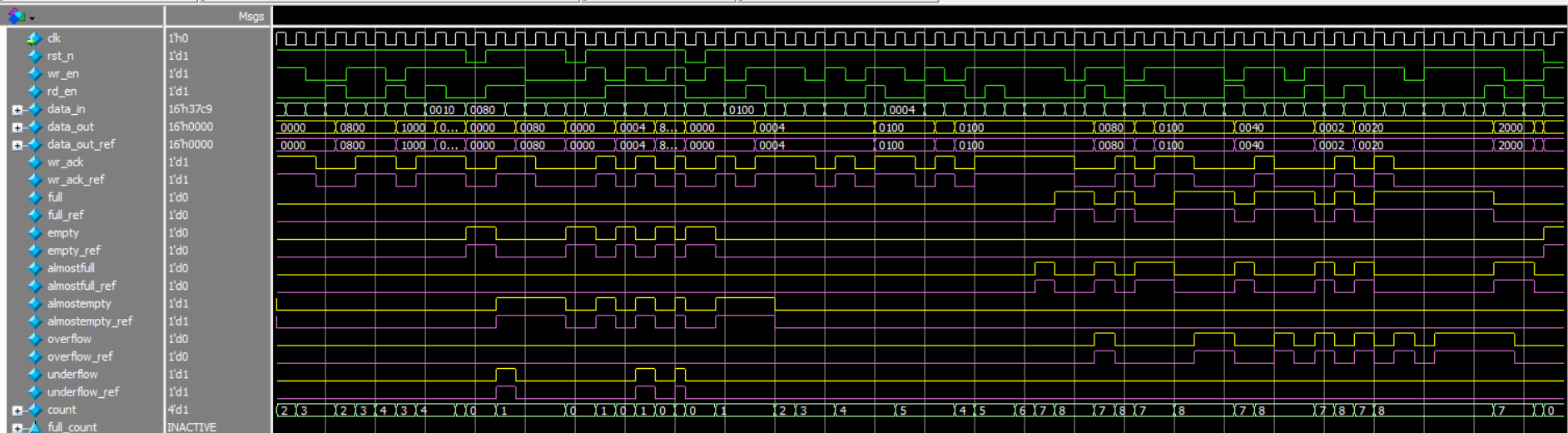
|  |  |
| --- | --- |
| Bugs report summary. | |
| 1. wr\_ack | reset when reset is activated |
| 1. overflow | reset when reset is activated |
| 1. underflow | Add instruction to make it sequential. |
| 1. underflow | reset when reset is activated |
| 1. data\_out | reset when reset is activated |
| 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

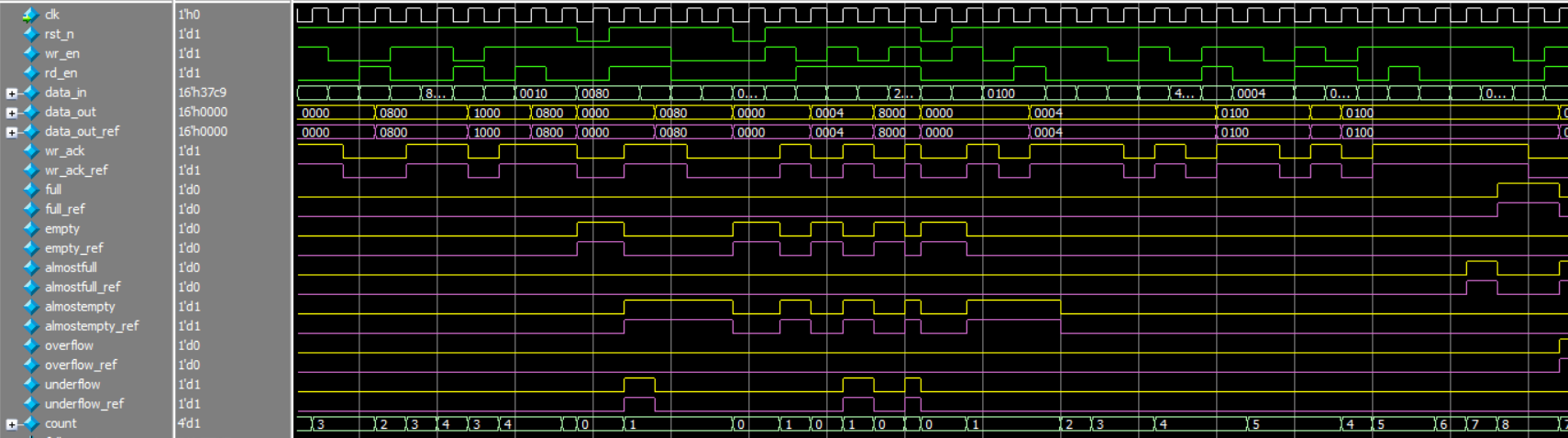
/// Big Picture for assertion shows that they are activated///



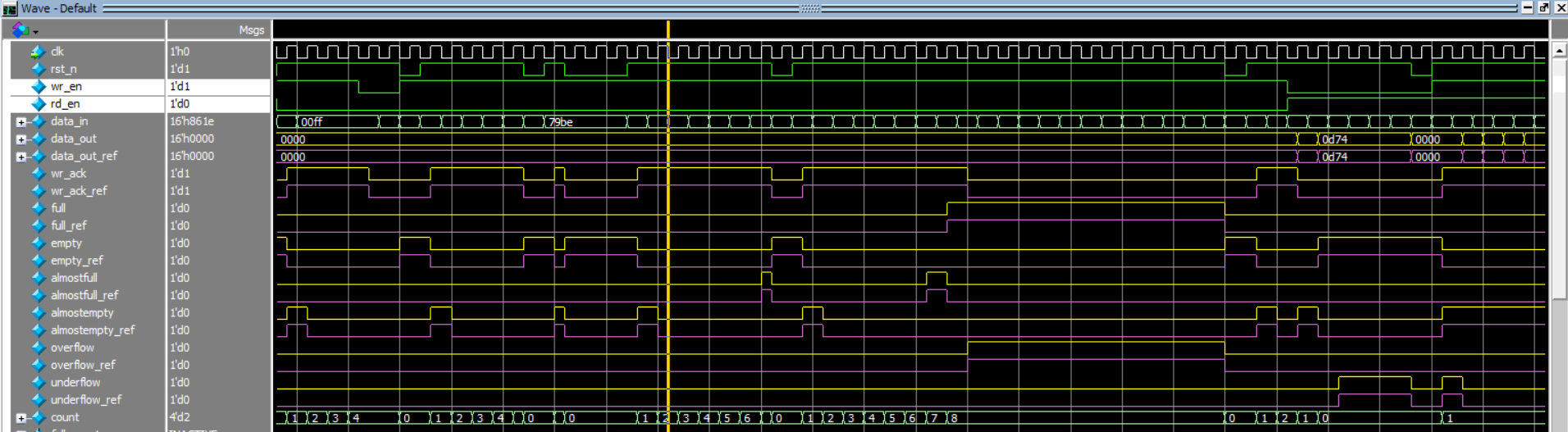
/// Big Picture for signals ///



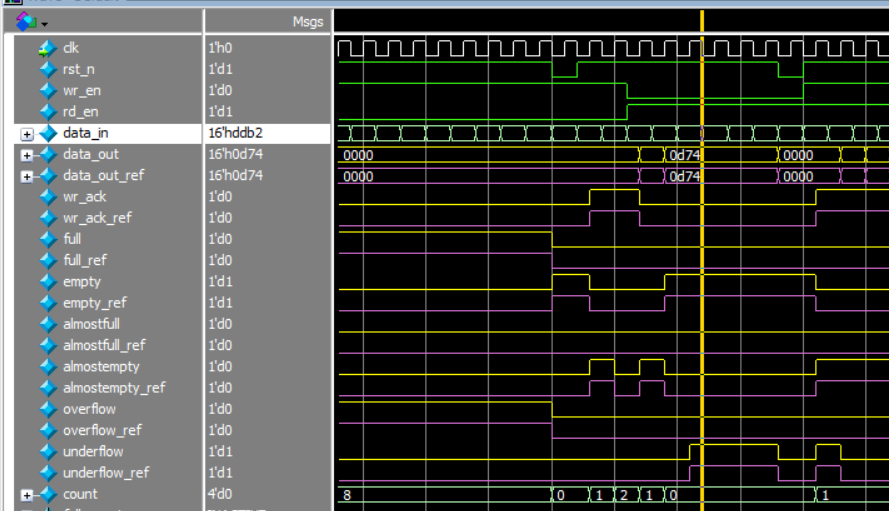
// Constraint CON\_W\_R\_DIST ///



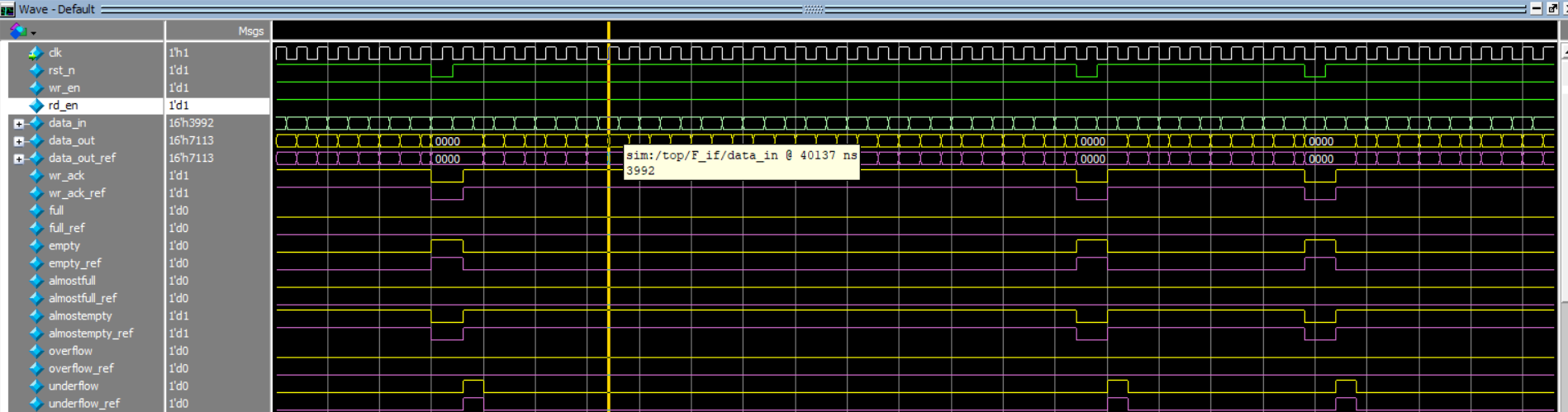
/// Constraint MY\_CON\_ONLEY\_W ///



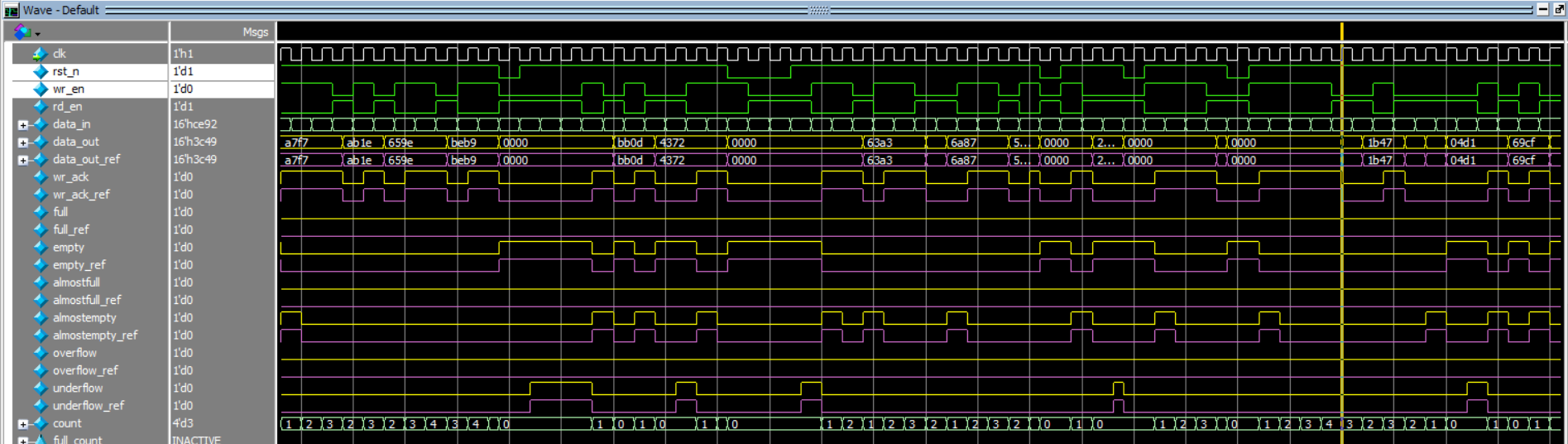
/// Constraint MY\_CON\_ONLEY\_R ///



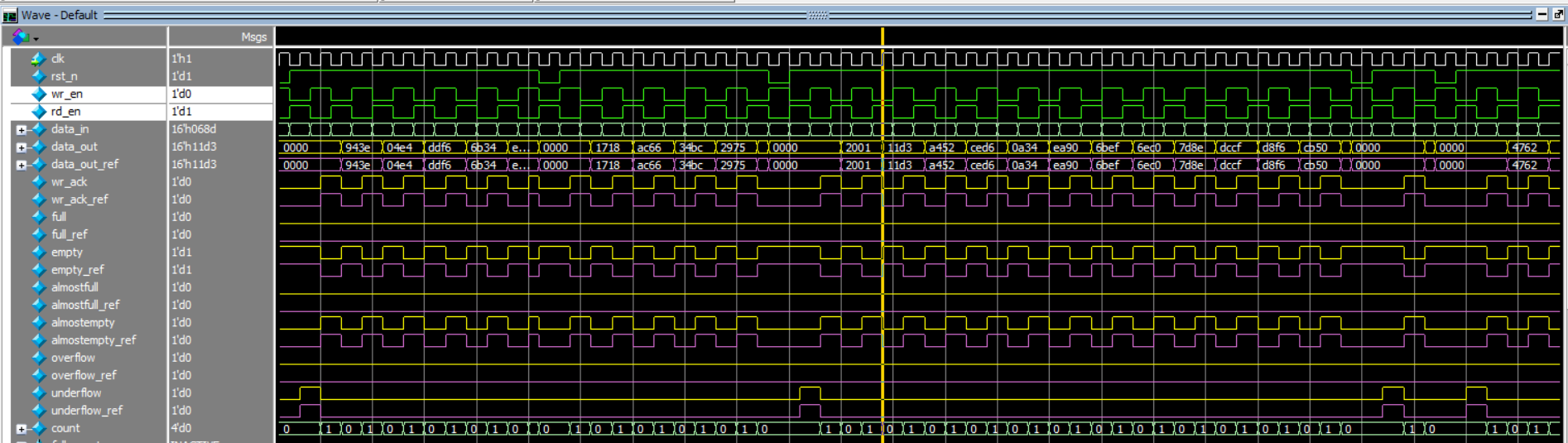
/// Constraint MY\_CON\_BOTH\_ACTIVE ///



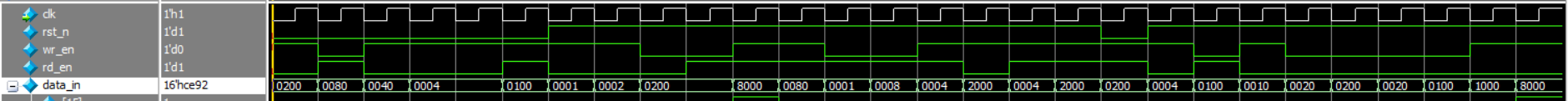
/// Constraint MY\_CON\_OPPSITE ///



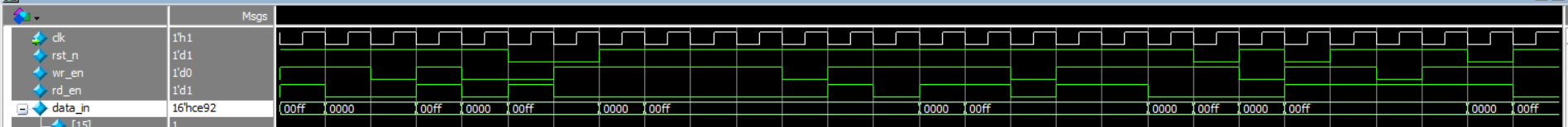
/// Constraint Toggle ///



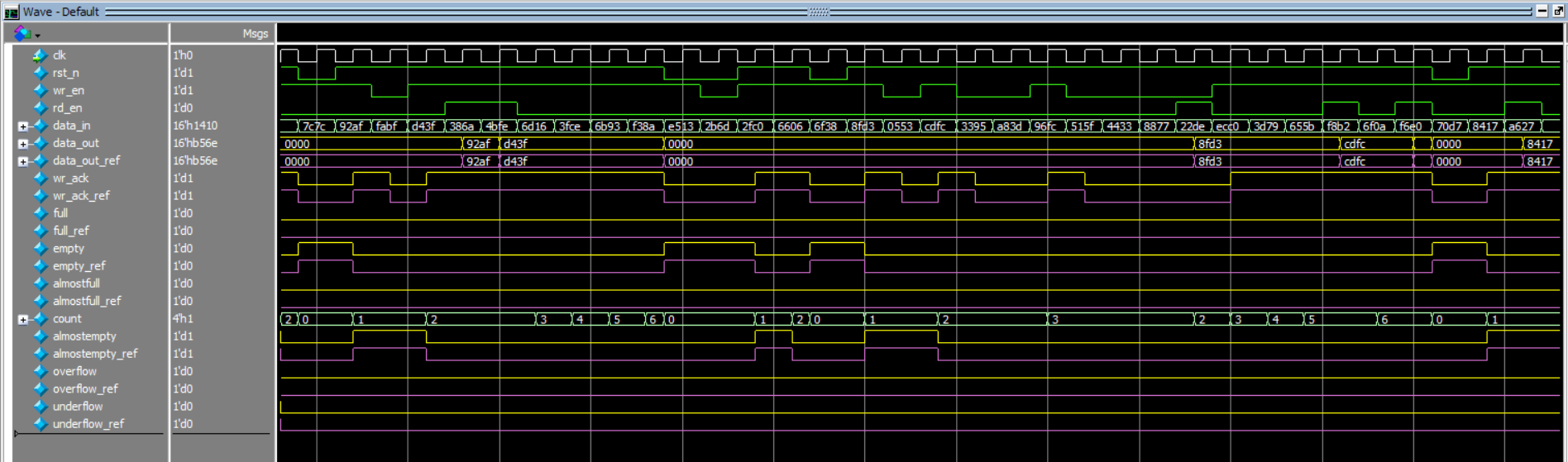
/// Constraint CON\_DATA\_OUT\_ONE\_BIT ///



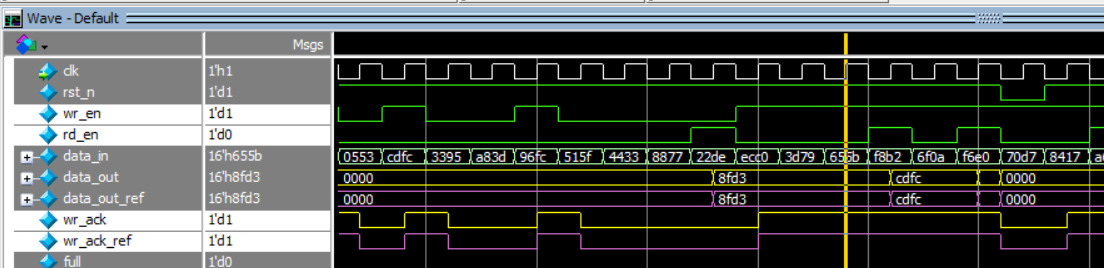
/// Constraint CON\_DATA\_OUT\_M\_Z ///



/// reset activate ///



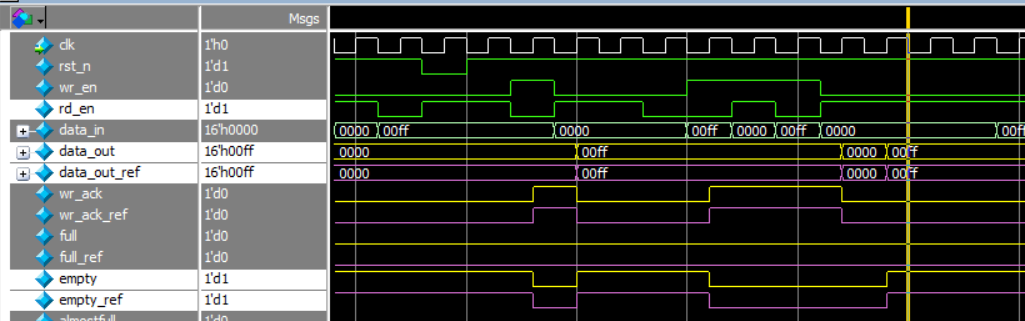
/// Write Acknowledge///



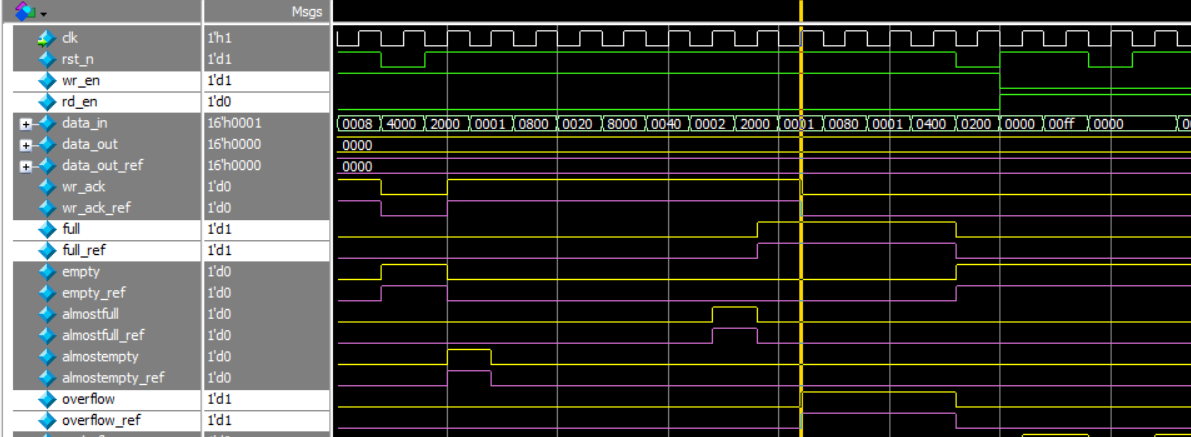
/// Write until gets full///



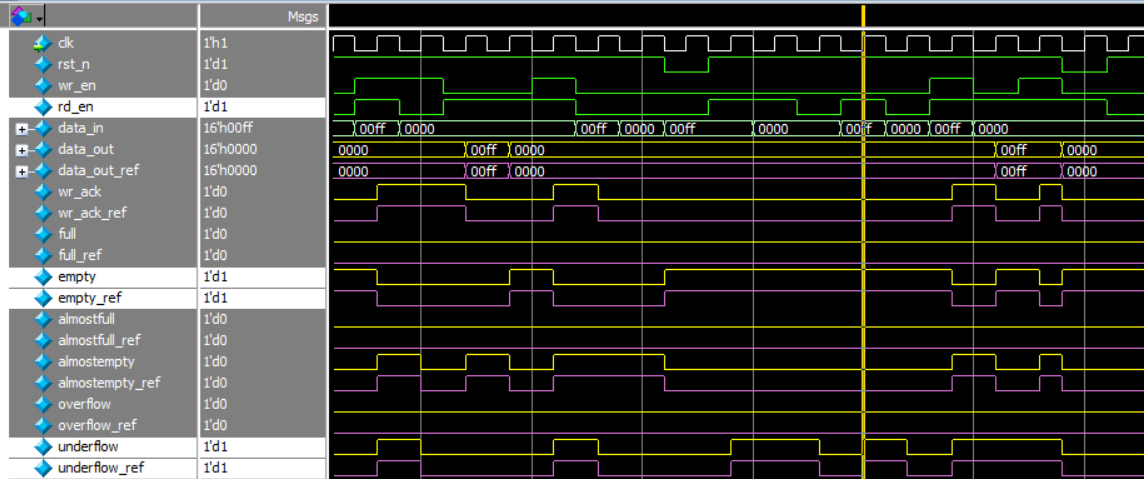
/// Read until gets empty///



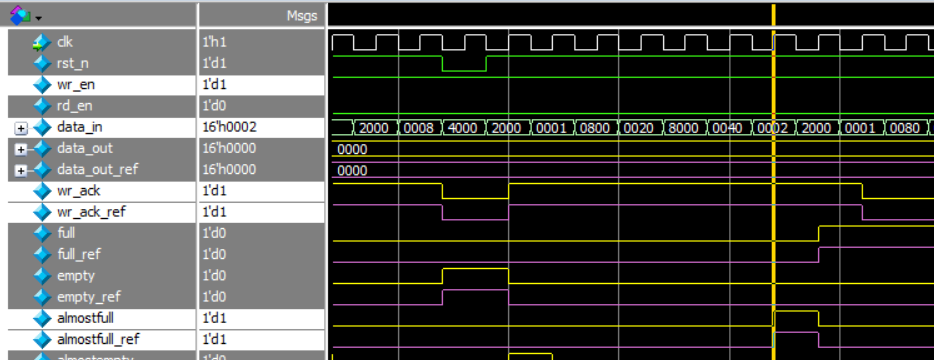
/// Overflow///



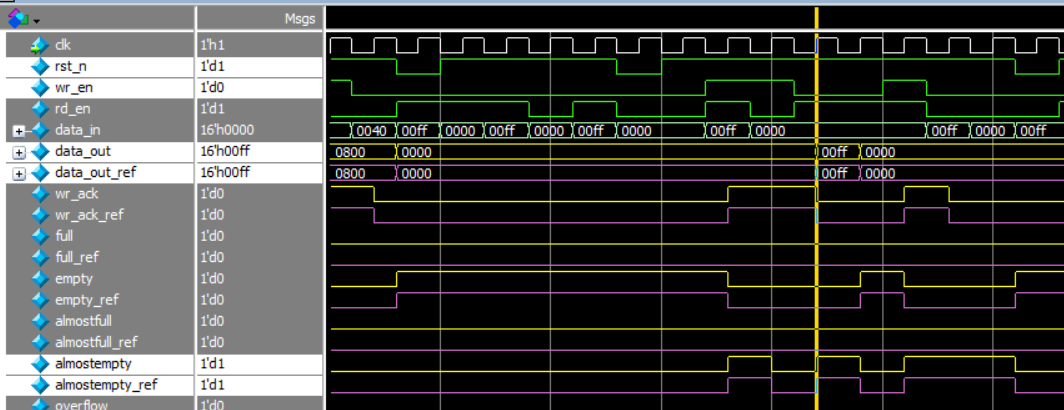
/// Underflow///



/// almost\_full///



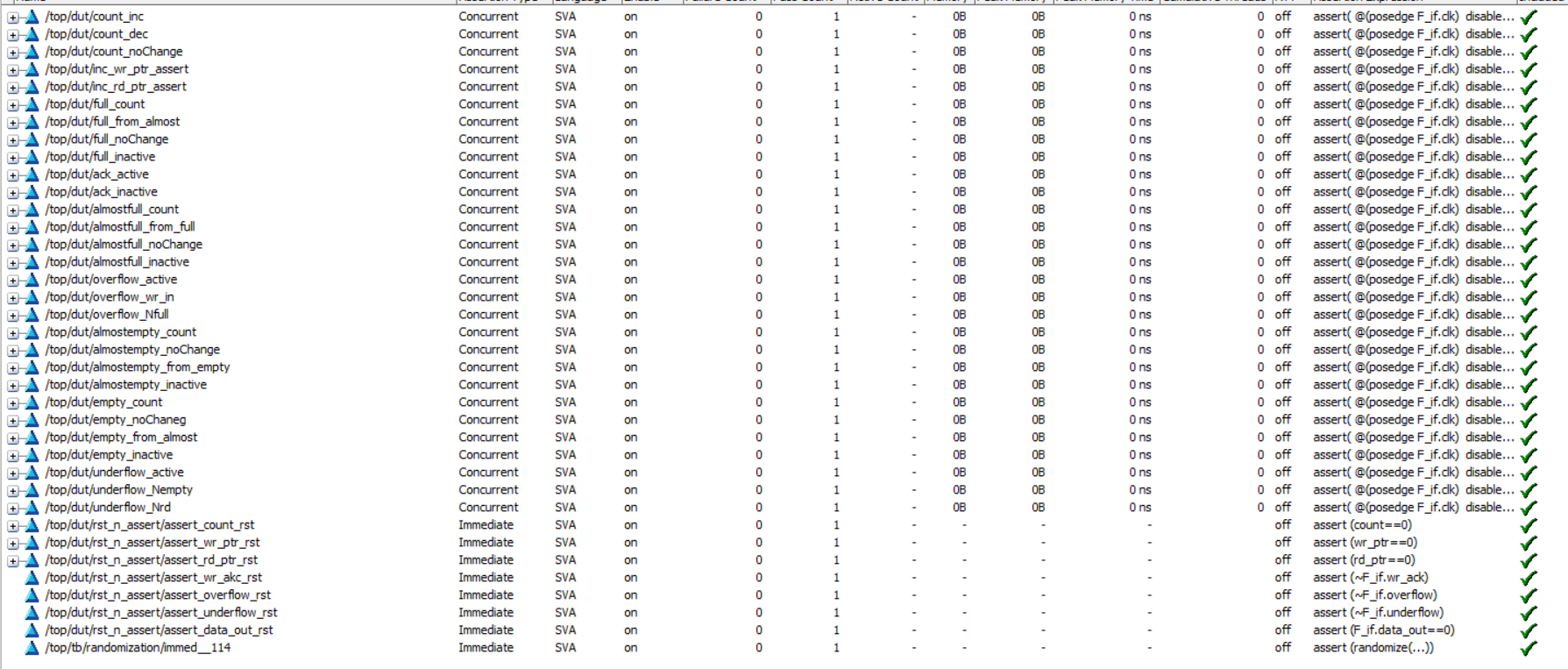
/// almost\_empty///



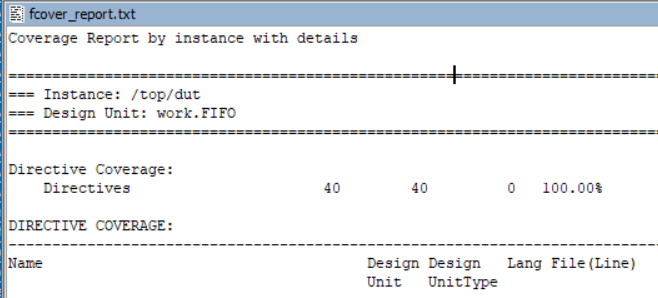
**/// Summary at the end of simulation ///**

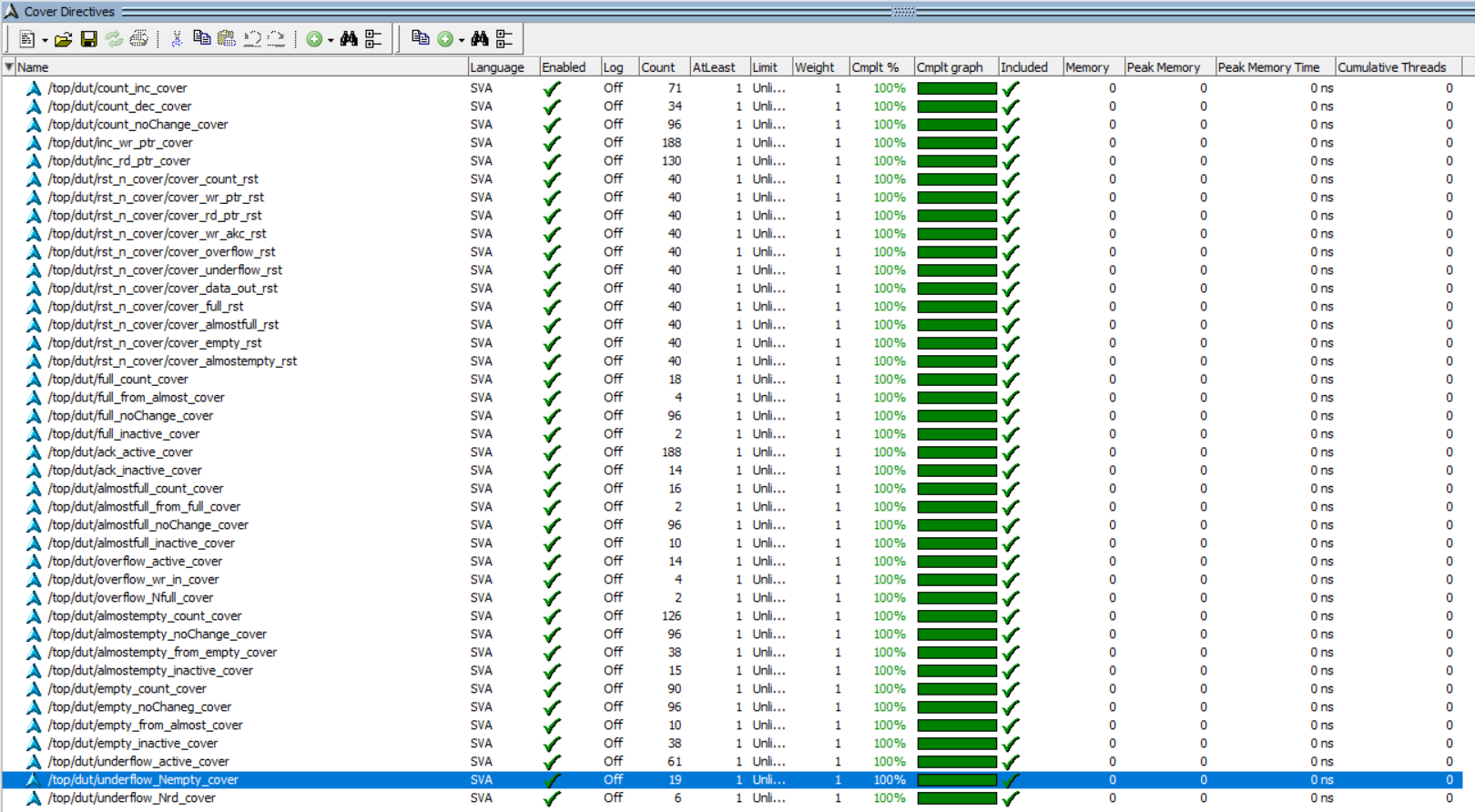


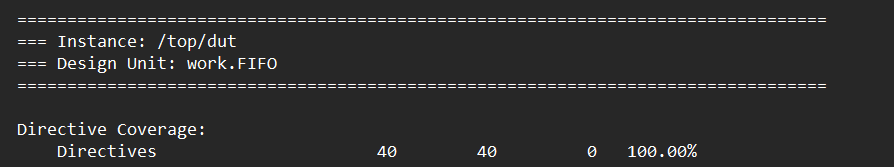
**/// Assertion ///**



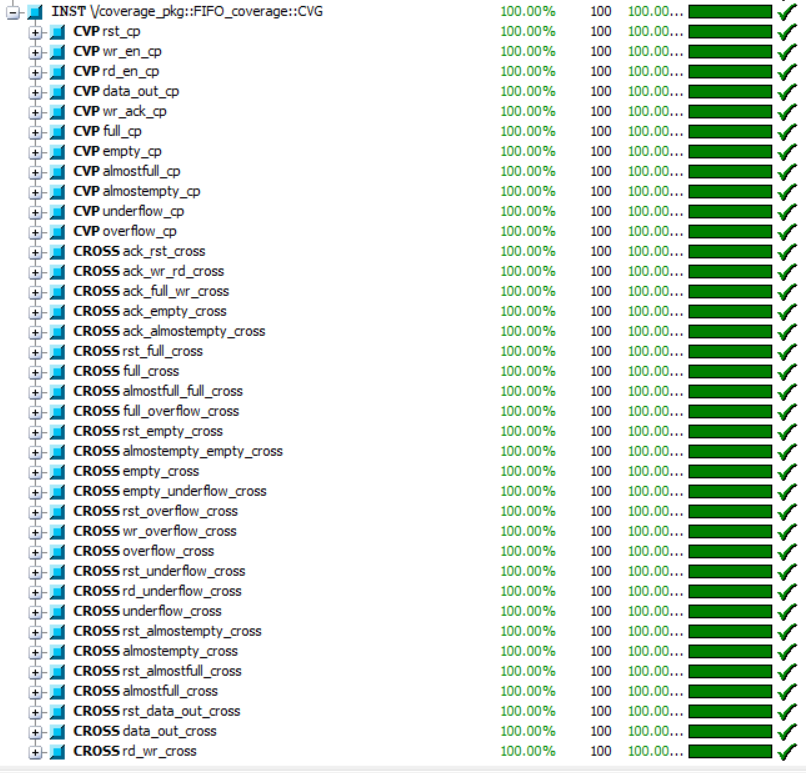
**/// Coverage directive ///**

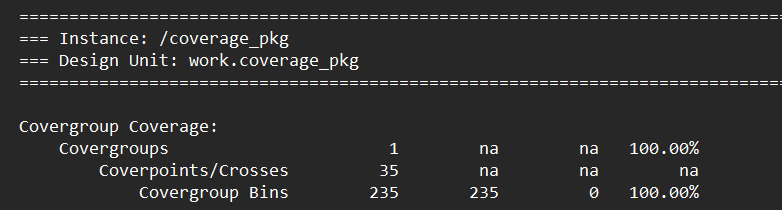
****

****

****

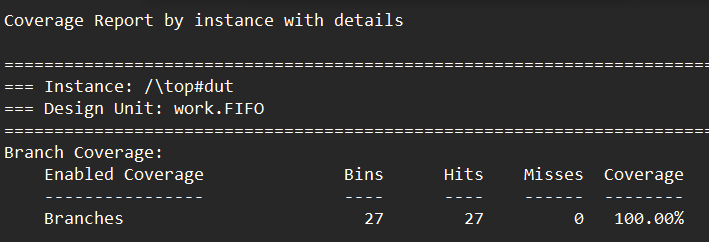
**/// Covergroup ///**

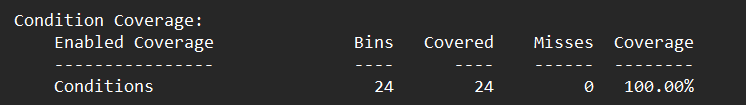


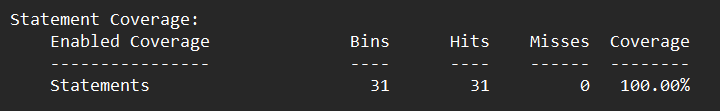


**/// Code Coverage Summary ///**

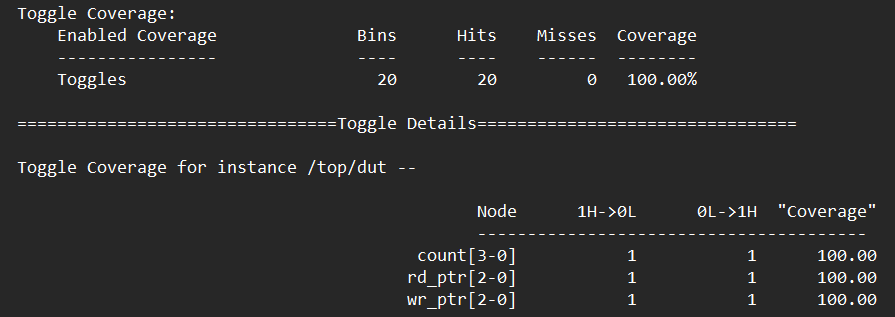








FIFO toggle for internal signal (wr\_ptr, rd\_ptr and count)



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

