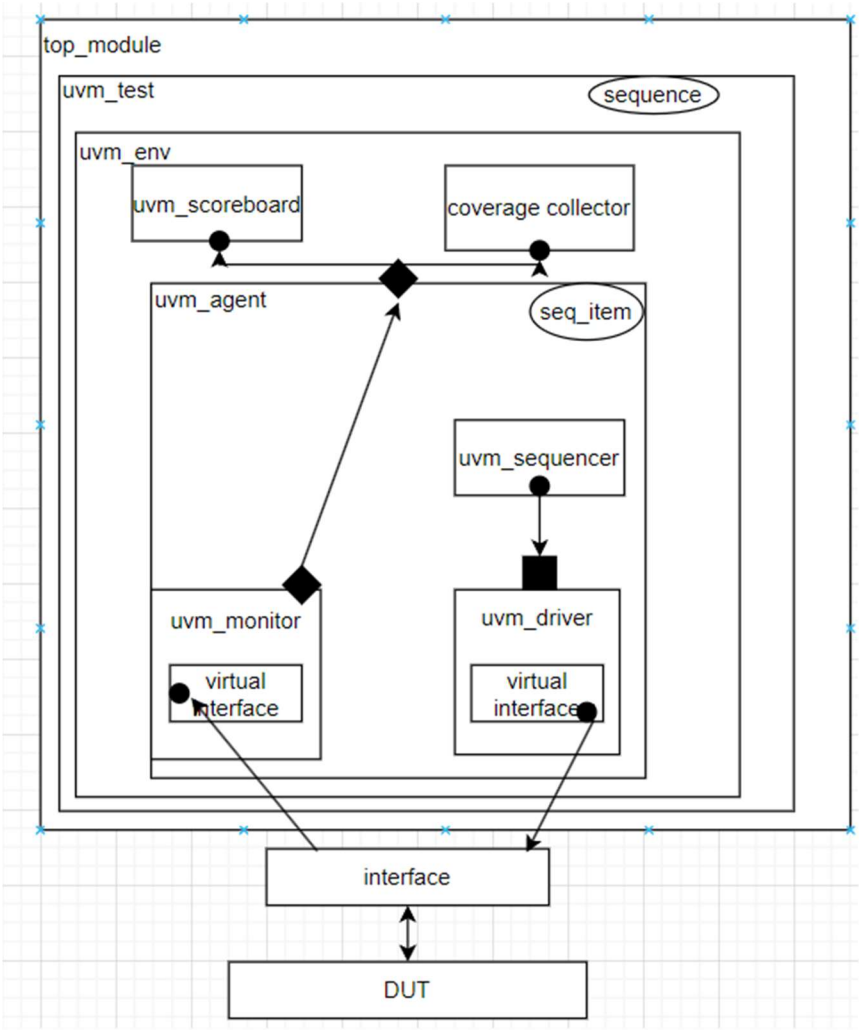
UVM_Project

Verification plan:

Label	Description	Stimulus Generation	Functional Coverage (Later)	Functionality Check
FIFO_1	When the reset is asserted,raise empty flag and lower other flags and counters	Directed at the start of the simulation	-	reset assertion
FIFO_2	wr_en=1, full=0 so, datain is written in fifo	randomized during simulation	cross coverage containing wr_en	checked using waveform
FIFO_3	rd_en=1, empty=0 so,dataout=data in fifo	randomized during simulation	cross coverage containing rd_en	checked using scoreboard
FIFO_4	wr_en=1, full=1 so raise overflowa	randomized during simulation	cross coverage containing wr_en & overflow	overflow assertion
FIFO_5	rd_en=1, empty=1 so, raise underflow	randomized during simulation	cross coverage containing rd_en & empty	underflow assertion
FIFO_6	count==8 so, raise full flag	randomized during simulation	cross coverage containing full	full assertion
FIFO_7	count=7 so, raise almostfull flag	randomized during simulation	cross coverage containing almostempty	almostfull assertion
FIFO_8	coumt=0 so, raise empty flag	randomized during simulation	cross coverage containing empty	empty assertion
FIFO_9	count=1 so, raise almostempty flag	randomized during simulation	cross coverage containing almostempty	almostempty assertion
FIFO_10	wr_en=1, full=0 so, raise wr_ack	randomized during simulation	cross coverage containing wr_ack	wr_ack assertion

Drawing testbench using draw.io:



Top module generates clock, instantiates DUT and interface and runs Test. Test builds env which creates the agent and gets vif and pass it to config object. Agent gets config object and creates driver which drive signals through seq_item and sequencer to manage flow of data between sequence and driver and monitor passes data to be written in sequencer to start check and coverage

Coverage report:

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

Condition Coverage:
    Enabled Coverage              Bins   Covered    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Conditions                      24        24         0   100.00%

================================Condition Details=======================

Condition Coverage for instance /top/DUT --
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                      34        34         0   100.00%

================================Statement Details=======================

Statement Coverage for instance /top/DUT --
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                        106       106         0   100.00%

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

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

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

Covergroup Coverage:
    Covergroups                     1        na        na   100.00%
        Coverpoints/Crosses        16        na        na        na
            Covergroup Bins        58        58         0   100.00%
```

```
TOTAL DIRECTIVE COVERAGE: 100.00%  COVERS: 14

ASSERTION RESULTS:
--------------------------------------------------------------------
Name                File(Line)                   Failure      Pass
                                                 Count        Count
--------------------------------------------------------------------
/top/DUT/fifo_sva_inst/rst_assertion
                    fifo_assertions.sv(72)             0          1
/top/DUT/fifo_sva_inst/full_assertion
                    fifo_assertions.sv(73)             0          1
/top/DUT/fifo_sva_inst/almostfull_assertion
                    fifo_assertions.sv(74)             0          1
/top/DUT/fifo_sva_inst/empty_assertion
                    fifo_assertions.sv(75)             0          1
/top/DUT/fifo_sva_inst/almostempty_assertion
                    fifo_assertions.sv(76)             0          1
/top/DUT/fifo_sva_inst/overflow_assertion
                    fifo_assertions.sv(77)             0          1
/top/DUT/fifo_sva_inst/underflow_assertion
                    fifo_assertions.sv(78)             0          1
/top/DUT/fifo_sva_inst/wr_ack_assertion
                    fifo_assertions.sv(79)             0          1
/top/DUT/fifo_sva_inst/counters_write_assertion
                    fifo_assertions.sv(80)             0          1
/top/DUT/fifo_sva_inst/counters_read_assertion
                    fifo_assertions.sv(81)             0          1
/top/DUT/fifo_sva_inst/wr_rd_both_empty_assertion
                    fifo_assertions.sv(82)             0          1
/top/DUT/fifo_sva_inst/wr_rd_both_full_assertion
                    fifo_assertions.sv(83)             0          1
/top/DUT/fifo_sva_inst/count_inc_assertion
                    fifo_assertions.sv(84)             0          1
/top/DUT/fifo_sva_inst/count_dec_assertion
                    fifo_assertions.sv(85)             0          1
/sequence_pkg/write_read_Sequence/body/#ublk#50851543#61/immed__64
                    fifo_sequence.sv(64)               0          1
```

Do file:

```
vlib work
vlog -f src_files.txt +cover -covercells
vsim -voptargs=+acc work.top -cover
add wave /top/f_if/*
add wave /top/DUT/mem
add wave /top/DUT/wr_ptr
add wave /top/DUT/rd_ptr
add wave /top/DUT/count
coverage save top.ucdb -onexit
run -all
```

Src_files.txt:

```
fifo_interface.sv
FIFO.sv
fifo_seq_item.sv
fifo_sequencer.sv
fifo_sequence.sv
fifo_config_obj.sv
fifo_driver.sv
fifo_monitor.sv
fifo_agent.sv
fifo_cvg_collector.sv
fifo_scoreboard.sv
fifo_env.sv
fifo_test.sv
fifo_assertions.sv
fifo_top.sv
```

Interface:

```
interface fifo_if (clk) ;
parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;
input clk;
logic [FIFO_WIDTH-1:0] data_in;
logic rst_n, wr_en, rd_en;
logic [FIFO_WIDTH-1:0] data_out;
logic wr_ack, overflow;
logic full, empty, almostfull, almostempty, underflow;
modport DUT (
input data_in,clk, rst_n, wr_en, rd_en,
output data_out,wr_ack,overflow,full,empty,almostfull,almostempty,underflow
);
endinterface
```

Design (bugs mentioned as comments) :

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

```
        overflow<=0;    //overflow=0 as fifo is not full since count<fifo_depth
    end
    else begin
        wr_ack <= 0;
        if (full && wr_en) //&& not &
            overflow <= 1;
        else overflow <= 0;
    end
end

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        rd_ptr <= 0; underflow<=0;   //underflow is sequential
    end
    else if (rd_en && count != 0) begin
        data_out <= mem[rd_ptr];
        rd_ptr <= rd_ptr + 1;
        underflow<=0; //underflow=0 since fifo is not empty as count !=0
    end
    else begin
        if(empty && rd_en)  //added if else of the sequential output underflow
            underflow<=1;
        else underflow<=0;
    end
end

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        count <= 0;
    end
    else begin   //count value wasn't changing if rd_en==1 && wr_en==1
        if  ( ({wr_en, rd_en} == 2'b10) && !full)
        count<=count+1;
        else if ( ({wr_en, rd_en} == 2'b01) && !empty)
```

```
                count<=count-1;
            else if ( wr_en && rd_en && empty)
                count<=count+1;
            else if ( wr_en && rd_en && full)
                count<=count-1;
        end
    end

    assign full = (count == FIFO_DEPTH)? 1 : 0;
    assign empty = (count == 0)? 1 : 0;
    //assign underflow = (empty && rd_en)? 1 : 0; underflow is sequential
    assign almostfull = (count == FIFO_DEPTH-1)? 1 : 0; //almostfull is @count=7 not 6
    assign almostempty = (count == 1)? 1 : 0;
```

Seq_item:

```
package fifo_seq_item_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
class MySequenceItem extends uvm_sequence_item;
`uvm_object_utils(MySequenceItem);
rand logic [15:0] data_in;
rand logic rst_n, wr_en, rd_en;
logic [15:0] data_out;
logic wr_ack, overflow;
logic full, empty, almostfull, almostempty, underflow;
    function new(string name="MySequenceItem");
        super.new(name);
    endfunction //new()

    function string convert2string();
 return $sformatf ("%s,rst_n=0b%0b,wr_en=0b%0b,rd_en=0b%0b,data_in=0b%0b,wr_ack=0b%0b,overflow=0b%0b,full=0b%0b,empty=0b%0b,almostfull=%s,al
    endfunction

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

    constraint reset_con{
        rst_n dist {1:/90,0:/10};
    }
    constraint write_en {
        wr_en dist {1:/70,0:/30};
    }
    constraint read_en {
        rd_en dist {1:/30,0:/70};
    }
endclass
endpackage
```

Sequencer:

```
package fifo_sequencer_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
import fifo_seq_item_pkg::*;
class MySequencer extends uvm_sequencer #(MySequenceItem);
`uvm_component_utils(MySequencer);
    function new(string name = "MySequencer",uvm_component parent=null);
        super.new(name,parent);
    endfunction //new()
endclass
endpackage
```

Sequence:

```
package sequence_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
import fifo_seq_item_pkg::*;
class reset_Sequence extends uvm_sequence #(MySequenceItem);
`uvm_object_utils(reset_Sequence);
MySequenceItem item;
    function new(string name = "MySequence");
        super.new(name);
    endfunction //new()

    task body();
            item = MySequenceItem::type_id::create("item") ;
            start_item(item);
            item.rst_n=0; item.wr_en=0; item.rd_en=0; item.data_in=0;
            finish_item(item);
    endtask
endclass

class write_Sequence extends uvm_sequence #(MySequenceItem);
`uvm_object_utils(write_Sequence);
MySequenceItem item;
    function new(string name = "MySequence");
        super.new(name);
    endfunction //new()
    task body();
       repeat(50) begin
            item = MySequenceItem::type_id::create("item") ;
            start_item(item);
            item.rst_n=1; item.wr_en=1; item.rd_en=0; item.data_in=$random;
            finish_item(item);
        end
    endtask
endclass

class read_Sequence extends uvm_sequence #(MySequenceItem);
`uvm_object_utils(read_Sequence);
```

```
MySequenceItem item;
    function new(string name = "MySequence");
        super.new(name);
    endfunction //new()
     task body();
        repeat(50) begin
            item = MySequenceItem::type_id::create("item") ;
            start_item(item);
            item.rst_n=1; item.wr_en=0; item.rd_en=1;
            finish_item(item);
        end
    endtask
endclass

class write_read_Sequence extends uvm_sequence #(MySequenceItem);
`uvm_object_utils(write_read_Sequence);
MySequenceItem item;
    function new(string name = "MySequence");
        super.new(name);
    endfunction //new()
     task body();
        repeat(1000) begin
            item = MySequenceItem::type_id::create("item") ;
            start_item(item);
            assert(item.randomize());
            finish_item(item);
        end
    endtask
endclass
endpackage
```

Config_obj:

```
package fifo_config_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh";
class fifo_config extends uvm_object;
`uvm_object_utils(fifo_config);
virtual fifo_if fifo_vif;
    function new(string name="fifo_config");
        super.new(name);
    endfunction //new()
endclass
endpackage
```

Driver:

```
package fifo_driver_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh";
import fifo_config_pkg::*;   ///
import fifo_seq_item_pkg::*;
class fifo_driver extends uvm_driver #(MySequenceItem);
`uvm_component_utils(fifo_driver)
virtual fifo_if fifo_vif;
fifo_config fifo_cfg;
MySequenceItem stim_seq_item;

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

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever begin
            stim_seq_item=MySequenceItem::type_id::create("stim_seq_item");
            seq_item_port.get_next_item(stim_seq_item);
            fifo_vif.rst_n=stim_seq_item.rst_n; fifo_vif.wr_en=stim_seq_item.wr_en;
            fifo_vif.rd_en=stim_seq_item.rd_en; fifo_vif.data_in=stim_seq_item.data_in;
            @(negedge fifo_vif.clk);
            seq_item_port.item_done();
            `uvm_info("run_phase",stim_seq_item.convert2string_stimulus(),UVM_HIGH)
        end
    endtask
endclass
endpackage
```

Monitor:

```
package monitor_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
import fifo_seq_item_pkg::*;
class fifo_monitor extends uvm_monitor;
`uvm_component_utils(fifo_monitor)
virtual fifo_if fifo_vif;
MySequenceItem rsp_seq_item;
uvm_analysis_port #(MySequenceItem) mon_ap;

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

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

   task run_phase(uvm_phase phase);
    super.run_phase(phase);
    forever begin
        rsp_seq_item=MySequenceItem::type_id::create("rsp_seq_item");
        @(negedge fifo_vif.clk);
        rsp_seq_item.rst_n=fifo_vif.rst_n; rsp_seq_item.wr_en=fifo_vif.wr_en; rsp_seq_item.rd_en=fifo_vif.rd_en;
        rsp_seq_item.data_in=fifo_vif.data_in; rsp_seq_item.overflow=fifo_vif.overflow; rsp_seq_item.underflow=fifo_vif.underflow;
        rsp_seq_item.full=fifo_vif.full; rsp_seq_item.empty=fifo_vif.empty; rsp_seq_item.almostfull=fifo_vif.almostfull;
        rsp_seq_item.almostempty=fifo_vif.almostempty; rsp_seq_item.wr_ack=fifo_vif.wr_ack;
        mon_ap.write(rsp_seq_item);
        `uvm_info("run_phase",rsp_seq_item.convert2string(),UVM_HIGH)
    end
   endtask
endclass
endpackage
```

Agent:

```
package agent_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
import fifo_sequencer_pkg::*;
import fifo_driver_pkg::*;
import monitor_pkg::*;
import fifo_config_pkg::*;
import fifo_seq_item_pkg::*;
class fifo_agent extends uvm_agent;
`uvm_component_utils(fifo_agent)
MySequencer sqr;
fifo_driver drv;
fifo_monitor mon;
fifo_config fifo_cfg;
uvm_analysis_port #(MySequenceItem) agt_ap;
    function new(string name= "fifo_agent",uvm_component parent=null);
        super.new(name,parent);
    endfunction

    function void build_phase(uvm_phase phase);
       super.build_phase(phase) ;
       if(!uvm_config_db #(fifo_config)::get(this,"","CFG",fifo_cfg))
       `uvm_fatal("build_phase","unable to get config obj");
       sqr=MySequencer::type_id::create("sqr",this);
       drv=fifo_driver::type_id::create("drv",this);
       mon=fifo_monitor::type_id::create("mon",this);
       agt_ap=new("agt_ap",this);
    endfunction

   function void connect_phase(uvm_phase phase);
   drv.fifo_vif=fifo_cfg.fifo_vif;
   mon.fifo_vif=fifo_cfg.fifo_vif;
   drv.seq_item_port.connect(sqr.seq_item_export);
   mon.mon_ap.connect(agt_ap);
   endfunction
endclass
endpackage
```

Coverage collector:

```
package cvg_collector_pkg;
import fifo_seq_item_pkg::*;
import fifo_driver_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class fifo_coverage extends uvm_component;
`uvm_component_utils(fifo_coverage)
uvm_analysis_export #(MySequenceItem) cov_export;
uvm_tlm_analysis_fifo #(MySequenceItem) cov_fifo;
MySequenceItem seq_item_cov;
covergroup g1 () ;
rd_en_cp: coverpoint seq_item_cov.rd_en;
wr_en_cp: coverpoint seq_item_cov.wr_en;
overflow_cp: coverpoint seq_item_cov.overflow;
underflow_cp: coverpoint seq_item_cov.underflow;
full_cp: coverpoint seq_item_cov.full;
empty_cp: coverpoint seq_item_cov.empty;
almostfull_cp: coverpoint seq_item_cov.almostfull;
almostempty_cp: coverpoint seq_item_cov.almostempty;
wr_ack_cp: coverpoint seq_item_cov.wr_ack;

wr_rd_wrack: cross rd_en_cp,wr_ack_cp,wr_en_cp{
    ignore_bins wren_ack = binsof (wr_en_cp) intersect {0} && binsof (wr_ack_cp);
}
wr_rd_overflow: cross wr_en_cp,rd_en_cp,overflow_cp{
    ignore_bins wren_overflow = binsof (wr_en_cp) intersect {0} && binsof (overflow_cp);
}
wr_rd_full: cross wr_en_cp,rd_en_cp,full_cp{
    ignore_bins rden_full = binsof (rd_en_cp) intersect {1} && binsof (full_cp);
}
wr_rd_empty: cross wr_en_cp,rd_en_cp,empty_cp;
wr_rd_almostfull: cross wr_en_cp,rd_en_cp,almostfull_cp;
wr_rd_almostempty: cross wr_en_cp,rd_en_cp,almostempty_cp;
wr_rd_underflow: cross wr_en_cp,rd_en_cp,underflow_cp{
    ignore_bins rden_underflow = binsof (rd_en_cp) intersect {0} && binsof (underflow_cp);
}
endgroup
```

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

Scoreboard:

```
package fifo_scoreboard_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh";
import fifo_seq_item_pkg::*;
import fifo_driver_pkg::*;
class fifo_scoreboard extends uvm_scoreboard;
`uvm_component_utils (fifo_scoreboard)
uvm_analysis_export #(MySequenceItem) sb_export;
uvm_tlm_analysis_fifo #(MySequenceItem) sb_fifo;
MySequenceItem seq_item_sb;
logic [15:0] data_out_ref;
integer errors_count=0;
integer correct_count=0;
logic [15:0] fifo_ref [$];
integer count=0;

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

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

    function void connect_phase (uvm_phase phase);
        super.connect_phase(phase);
        sb_export.connect (sb_fifo.analysis_export);
    endfunction

    task run_phase (uvm_phase phase);
        super.run_phase (phase);
        forever begin
            sb_fifo.get(seq_item_sb);
            ref_model(seq_item_sb);
            if (seq_item_sb.data_out!=data_out_ref) begin
```

```
`uvm_error ("run_phase",$sformatf("comparison failed, dut_out=%s and ref_model_out=0b%0b",seq_item_sb.convert2string(),data_out_ref));
                errors_count=errors_count+1;
            end
            else begin
                `uvm_info ("run_phase",$sformatf("correct out=%s",seq_item_sb.convert2string()),UVM_HIGH);
                correct_count=correct_count+1;
            end
        end
    endtask

    task ref_model(MySequenceItem seq_item_chk);
    if(!seq_item_chk.rst_n) begin
        count=0;
        fifo_ref.delete();
    end

    else begin
        if(seq_item_chk.wr_en==1 && seq_item_chk.rd_en==1 ) begin
            if(count==0) begin
                fifo_ref.push_back(seq_item_chk.data_in);
                count=count+1;
            end
            else if (count==8) begin
                data_out_ref=fifo_ref.pop_front();
                count=count-1;
            end
            else begin
                fifo_ref.push_back(seq_item_chk.data_in);
                data_out_ref=fifo_ref.pop_front();
                end
        end
else if (seq_item_chk.wr_en==1 && seq_item_chk.rd_en==0 && count<8) begin   //write case
        fifo_ref.push_back(seq_item_chk.data_in);
```

```
    else if (seq_item_chk.wr_en==1 && seq_item_chk.rd_en==0 && count<8) begin   //write case
            fifo_ref.push_back(seq_item_chk.data_in);
            count=count+1;
    end

    else if (seq_item_chk.wr_en==0 && seq_item_chk.rd_en==1 && count>0) begin   //read case
            data_out_ref=fifo_ref.pop_front();
            count=count-1;
    end
        end
        endtask

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

Env:

```
package env_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"
import agent_pkg::*;
import cvg_collector_pkg::*;
import fifo_scoreboard_pkg::*;
class fifo_env extends uvm_env;
`uvm_component_utils (fifo_env)
fifo_agent agt;
fifo_coverage cov;
fifo_scoreboard sb;
    function new(string name="fifo_env",uvm_component parent=null);
        super.new (name,parent);
    endfunction //new()
    function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        agt=fifo_agent::type_id::create("agt",this);
        cov=fifo_coverage::type_id::create("cov",this);
        sb=fifo_scoreboard::type_id::create("sb",this);
    endfunction
    function void connect_phase (uvm_phase phase);
        agt.agt_ap.connect(sb.sb_export);
        agt.agt_ap.connect (cov.cov_export);
    endfunction
endclass
endpackage
```

Test:

```
package test_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh";
import env_pkg::*;
import fifo_config_pkg::*;
import fifo_driver_pkg::*;
import sequence_pkg::*;
import agent_pkg::*;
class fifo_test extends uvm_test;
`uvm_component_utils(fifo_test);
fifo_env env;
fifo_config fifo_cfg;
virtual fifo_if fifo_vif;
write_Sequence write_seq;
read_Sequence read_seq;
write_read_Sequence write_read_seq;
reset_Sequence reset_seq;
    function new(string name = "fifo_test",uvm_component parent=null);
        super.new(name,parent);
    endfunction
    function void build_phase(uvm_phase phase);
       super.build_phase(phase) ;
       env=fifo_env::type_id::create("env",this);
       fifo_cfg=fifo_config::type_id::create("fifo_cfg",this); /////
       write_seq=write_Sequence::type_id::create("write_seq",this);
       read_seq=read_Sequence::type_id::create("read_seq",this);
       write_read_seq=write_read_Sequence::type_id::create("write_read_seq",this);
       reset_seq=reset_Sequence::type_id::create("reset_seq",this);
       if(!uvm_config_db #(virtual fifo_if)::get(this,"","fifo_if",fifo_cfg.fifo_vif))
        `uvm_fatal ("build_phase","test - unable to get configuration object")
        uvm_config_db #(fifo_config)::set(this,"*","CFG",fifo_cfg);
    endfunction
    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        phase.raise_objection(this);
        `uvm_info("run_phase","reset asserted",UVM_LOW)
        reset_seq.start(env.agt.sqr);
```

```
        `uvm_info("run_phase","reset DEasserted",UVM_LOW)
        `uvm_info("run_phase","write asserted",UVM_LOW)
        write_seq.start(env.agt.sqr);    //******************
        `uvm_info("run_phase","write deasserted",UVM_LOW)
        `uvm_info("run_phase","read asserted",UVM_LOW)
        read_seq.start(env.agt.sqr);
        `uvm_info("run_phase","read deasserted",UVM_LOW)
        `uvm_info("run_phase","write_read asserted",UVM_LOW)
        write_read_seq.start(env.agt.sqr);
        `uvm_info("run_phase","write_read deasserted",UVM_LOW)
        `uvm_info("run_phase","stim gen ended",UVM_LOW)
        phase.drop_objection(this);
    endtask
endclass
endpackage
```

Assertions:

```
module fifo_sva (fifo_if.DUT f_if);
logic clk, rst_n, wr_en, rd_en;
logic wr_ack, overflow, wr_ptr, rd_ptr;
logic full, empty, almostfull, almostempty, underflow;
logic [3:0] count;
assign clk=f_if.clk;
assign rst_n=f_if.rst_n;
assign wr_en=f_if.wr_en;
assign rd_en=f_if.rd_en;
assign wr_ptr=DUT.wr_ptr;
assign rd_ptr=DUT.rd_ptr;
assign count=DUT.count;

property rst_p;
@(posedge clk) (!rst_n) |-> ( !f_if.full && f_if.empty && !f_if.almostfull && !f_if.almostempty && !count && !rd_ptr && !wr_ptr );
endproperty

property full_p;
@(posedge clk) disable iff (!rst_n) (count==8) |-> (f_if.full);
endproperty

property almostfull_p;
@(posedge clk) disable iff (!rst_n) count==7 |-> (f_if.almostfull);
endproperty

property empty_p;
@(posedge clk) disable iff (!rst_n) count==0 |-> (f_if.empty) ;
endproperty

property almostempty_p;
@(posedge clk) disable iff (!rst_n) count==3'd1 |-> (f_if.almostempty);
endproperty

property overflow_p;
@(posedge clk) disable iff (!rst_n) (f_if.full && wr_en ) |=> (f_if.overflow);
endproperty
```

```
property underflow_p;
@(posedge clk) disable iff (!rst_n) (f_if.empty && rd_en) |=> (f_if.underflow);
endproperty

property wr_ack_p;
@(posedge clk) disable iff (!rst_n) (!f_if.full && wr_en) |=> (f_if.wr_ack);
endproperty

property counters_write_p;
@(posedge clk) disable iff (!rst_n) (!f_if.full && wr_en) |=> ( (wr_ptr==$past(wr_ptr)+1'b1)  );
endproperty

property counters_read_p;
@(posedge clk) disable iff (!rst_n) (!f_if.empty && rd_en)  |=> ( (rd_ptr==$past(rd_ptr)+1'b1)  );
endproperty

property wr_rd_both_empty;
@(posedge clk) disable iff (!rst_n) (wr_en && rd_en && f_if.empty) |=> ((f_if.wr_ack) && (wr_ptr==$past(wr_ptr)+1'b1) );
endproperty

property wr_rd_both_full;
@(posedge clk) disable iff (!rst_n) (wr_en && rd_en && f_if.full) |=> ((!f_if.wr_ack) && (rd_ptr==$past(rd_ptr)+1'b1) );
endproperty

property count_inc;
@(posedge clk) disable iff (!rst_n) (wr_en && !rd_en && !f_if.full) |=> (count==$past(count)+1'b1);
endproperty

property count_dec;
@(posedge clk) disable iff (!rst_n) (!wr_en && rd_en && !f_if.empty) |=> (count==$past(count)-1'b1);
endproperty
```

```
71  rst_assertion: assert property (rst_p);
72  full_assertion: assert property (full_p);
73  almostfull_assertion: assert property (almostfull_p);
74  empty_assertion: assert property (empty_p);
75  almostempty_assertion: assert property (almostempty_p);
76  overflow_assertion: assert property (overflow_p);
77  underflow_assertion: assert property (underflow_p);
78  wr_ack_assertion: assert property (wr_ack_p);
79  counters_write_assertion: assert property (counters_write_p);
80  counters_read_assertion: assert property (counters_read_p);
81  wr_rd_both_empty_assertion: assert property (wr_rd_both_empty);
82  wr_rd_both_full_assertion: assert property (wr_rd_both_full);
83  count_inc_assertion: assert property (count_inc);
84  count_dec_assertion: assert property (count_dec);
85
86  rst_cover: cover property (rst_p);
87  full_cover: cover property (full_p);
88  almostfull_cover: cover property (almostfull_p);
89  empty_cover: cover property (empty_p);
90  almostempty_cover: cover property (almostempty_p);
91  overflow_cover: cover property (overflow_p);
92  underflow_cover: cover property (underflow_p);
93  wr_ack_cover: cover property (wr_ack_p);
94  counters_write_cover: cover property (counters_write_p);
95  counters_read_cover: cover property (counters_read_p);
96  wr_rd_both_empty_cover: cover property (wr_rd_both_empty);
97  wr_rd_both_full_cover: cover property (wr_rd_both_full);
98  count_inc_cover: cover property (count_inc);
99  count_dec_cover: cover property (count_dec);
100 endmodule
```

Top:

```
1  import uvm_pkg::*;
2  `include "uvm_macros.svh"
3  import test_pkg::*;
4  import env_pkg::*;
5  import fifo_driver_pkg::*;
6  import fifo_config_pkg::*;
7  module top ();
8  bit clk;
9  initial begin
10     clk=0;
11     forever begin
12         #2; clk=~clk;
13     end
14 end
15 fifo_if f_if(clk);
16 FIFO DUT (f_if);
17 bind FIFO fifo_sva fifo_sva_inst (f_if);
18 initial begin
19     uvm_config_db #(virtual fifo_if)::set(null,"uvm_test_top","fifo_if",f_if);
20     run_test("fifo_test");
21 end
22 endmodule
```

Questasim snippets:

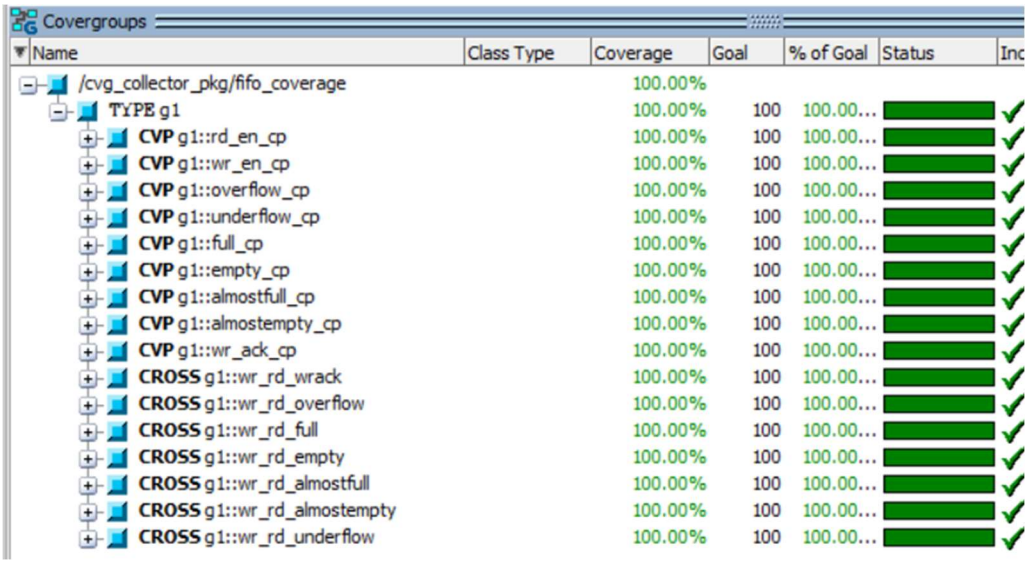
```
# UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(277) @ 0: reporter [Questa UVM] QUESTA_UVM-1.2.3
# UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(278) @ 0: reporter [Questa UVM]  questa_uvm::init(+struct)
# UVM_INFO @ 0: reporter [RNTST] Running test fifo_test...
# UVM_INFO fifo_test.sv(36) @ 0: uvm_test_top [run_phase] reset asserted
# UVM_INFO fifo_test.sv(38) @ 4: uvm_test_top [run_phase] reset DEasserted
# UVM_INFO fifo_test.sv(39) @ 4: uvm_test_top [run_phase] write asserted
# UVM_INFO fifo_test.sv(41) @ 204: uvm_test_top [run_phase] write deasserted
# UVM_INFO fifo_test.sv(42) @ 204: uvm_test_top [run_phase] read asserted
# UVM_INFO fifo_test.sv(44) @ 404: uvm_test_top [run_phase] read deasserted
# UVM_INFO fifo_test.sv(45) @ 404: uvm_test_top [run_phase] write_read asserted
# UVM_INFO fifo_test.sv(47) @ 4404: uvm_test_top [run_phase] write_read deasserted
# UVM_INFO fifo_test.sv(48) @ 4404: uvm_test_top [run_phase] stim gen ended
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 4404: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO fifo_scoreboard.sv(84) @ 4404: uvm_test_top.env.sb [report_phase] total successful transactions=1101
# UVM_INFO fifo_scoreboard.sv(85) @ 4404: uvm_test_top.env.sb [report_phase] total failed transactions=0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   15
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# [report_phase]     2
# [run_phase]     9
# ** Note: $finish    : C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 4404 ns  Iteration: 61  Instance: /top
```

Assertions

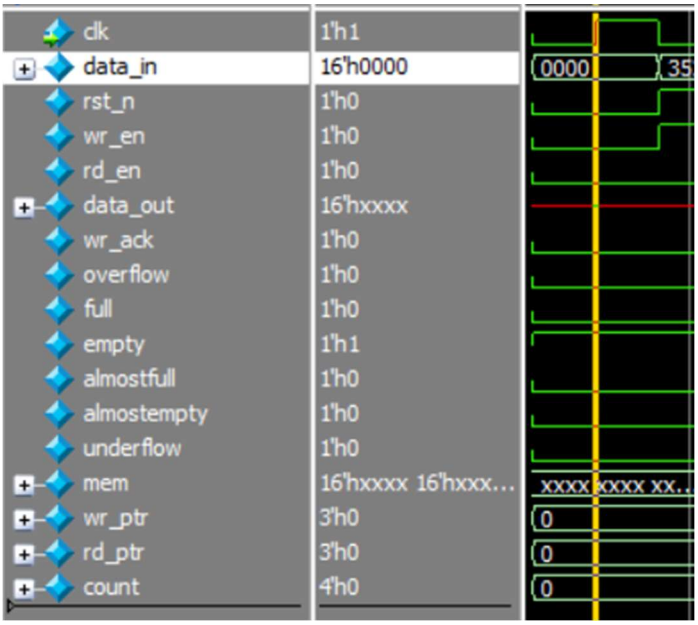
Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Count	Me
/uvm_pkg::uvm_reg_map::do_write/#ublk#215181159#1731/immed__173...	Immediate	SVA	on	0	0	-	
/uvm_pkg::uvm_reg_map::do_read/#ublk#215181159#1771/immed__177...	Immediate	SVA	on	0	0	-	
/sequence_pkg::write_read_Sequence::body/#ublk#50851543#61/immed_...	Immediate	SVA	on	0	1	-	
/top/DUT/fifo_sva_inst/rst_assertion	Concurrent	SVA	on	0	1	-	
/top/DUT/fifo_sva_inst/full_assertion	Concurrent	SVA	on	0	1	-	
/top/DUT/fifo_sva_inst/almostfull_assertion	Concurrent	SVA	on	0	1	-	
/top/DUT/fifo_sva_inst/empty_assertion	Concurrent	SVA	on	0	1	-	
/top/DUT/fifo_sva_inst/almostempty_assertion	Concurrent	SVA	on	0	1	-	
/top/DUT/fifo_sva_inst/overflow_assertion	Concurrent	SVA	on	0	1	-	
/top/DUT/fifo_sva_inst/underflow_assertion	Concurrent	SVA	on	0	1	-	
/top/DUT/fifo_sva_inst/wr_ack_assertion	Concurrent	SVA	on	0	1	-	
/top/DUT/fifo_sva_inst/counters_write_assertion	Concurrent	SVA	on	0	1	-	
/top/DUT/fifo_sva_inst/counters_read_assertion	Concurrent	SVA	on	0	1	-	
/top/DUT/fifo_sva_inst/wr_rd_both_empty_assertion	Concurrent	SVA	on	0	1	-	
/top/DUT/fifo_sva_inst/wr_rd_both_full_assertion	Concurrent	SVA	on	0	1	-	
/top/DUT/fifo_sva_inst/count_inc_assertion	Concurrent	SVA	on	0	1	-	
/top/DUT/fifo_sva_inst/count_dec_assertion	Concurrent	SVA	on	0	1	-	

Cover Directives

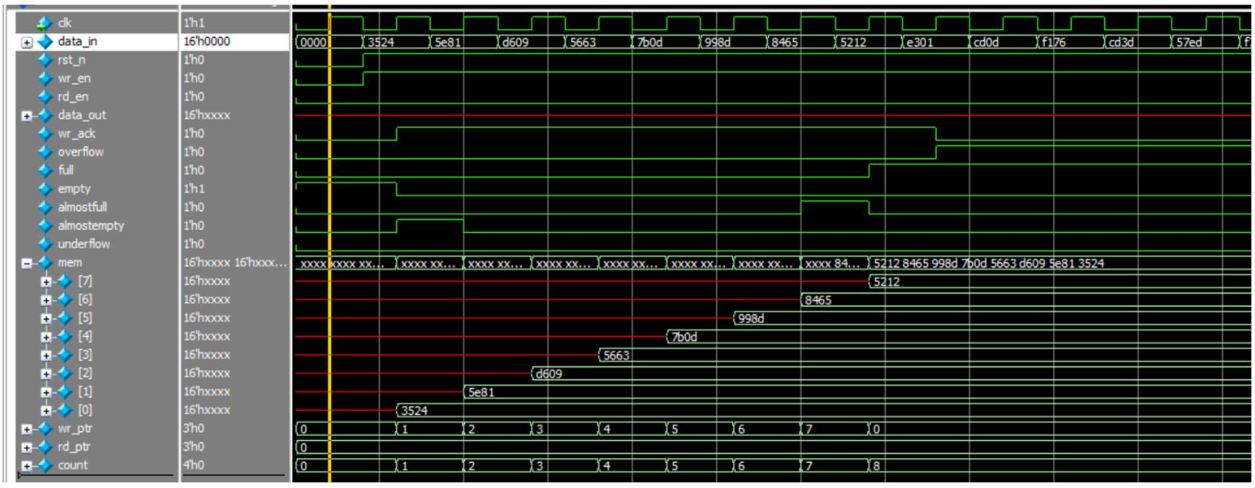
Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory
/top/DUT/fifo_sva_inst/rst_cover	SVA	✓	Off	95	1	Unli...	1	100%	✓		0
/top/DUT/fifo_sva_inst/full_cover	SVA	✓	Off	161	1	Unli...	1	100%	✓		0
/top/DUT/fifo_sva_inst/almostfull_cover	SVA	✓	Off	93	1	Unli...	1	100%	✓		0
/top/DUT/fifo_sva_inst/empty_cover	SVA	✓	Off	166	1	Unli...	1	100%	✓		0
/top/DUT/fifo_sva_inst/almostempty_co...	SVA	✓	Off	139	1	Unli...	1	100%	✓		0
/top/DUT/fifo_sva_inst/overflow_cover	SVA	✓	Off	118	1	Unli...	1	100%	✓		0
/top/DUT/fifo_sva_inst/underflow_cover	SVA	✓	Off	83	1	Unli...	1	100%	✓		0
/top/DUT/fifo_sva_inst/wr_ack_cover	SVA	✓	Off	499	1	Unli...	1	100%	✓		0
/top/DUT/fifo_sva_inst/counters_write_...	SVA	✓	Off	499	1	Unli...	1	100%	✓		0
/top/DUT/fifo_sva_inst/counters_read_c...	SVA	✓	Off	225	1	Unli...	1	100%	✓		0
/top/DUT/fifo_sva_inst/wr_rd_both_emp...	SVA	✓	Off	26	1	Unli...	1	100%	✓		0
/top/DUT/fifo_sva_inst/wr_rd_both_full_...	SVA	✓	Off	19	1	Unli...	1	100%	✓		0
/top/DUT/fifo_sva_inst/count_inc_cover	SVA	✓	Off	347	1	Unli...	1	100%	✓		0
/top/DUT/fifo_sva_inst/count_dec_cove...	SVA	✓	Off	80	1	Unli...	1	100%	✓		0



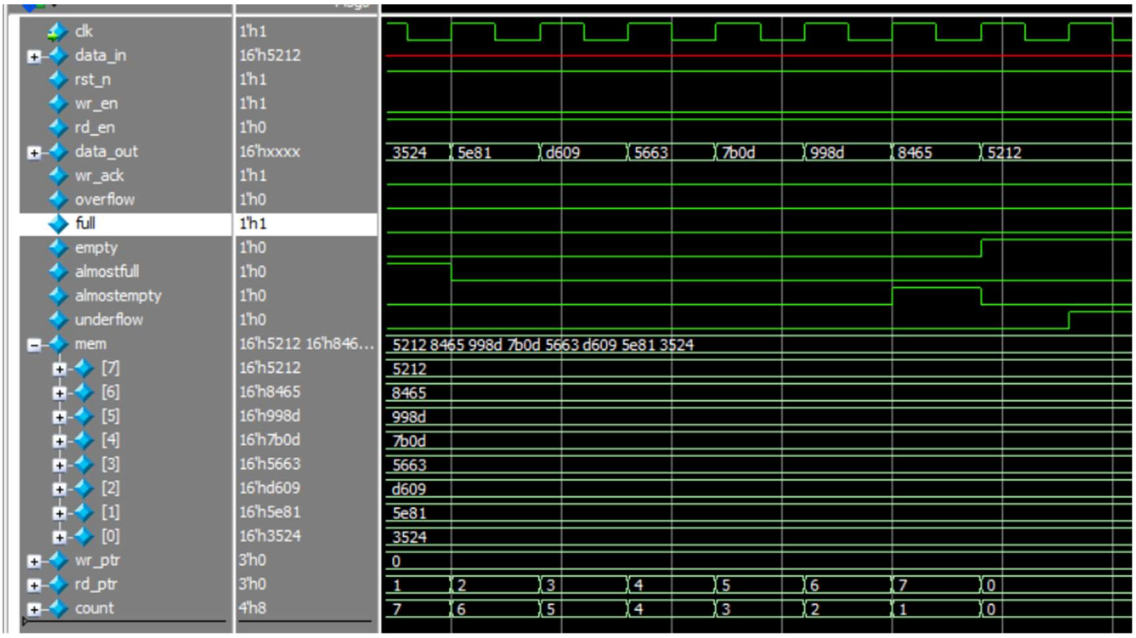
Reet sequence:



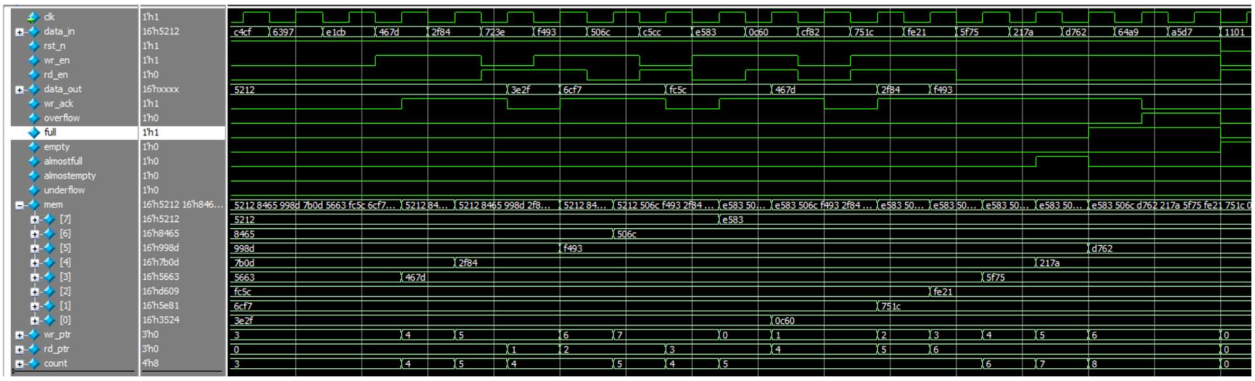
Write sequence:



Read sequence:



Write_read sequence:



Assertions table:

If reset is active, raise the empty flag and lower the other flags and counters	@(posedge clk) (!rst_n) |-> (!f_if.full && f_if.empty && !f_if.almostfull && !f_if.almostempty && !count && !rd_ptr && !wr_ptr)
If reset isn't active and we wrote in all fifo places (count==8), raise the full flag	@(posedge clk) disable iff (!rst_n) (count==8) |-> (f_if.full)
If reset isn't active, and there's one empty place in fifo, raise the	@(posedge clk) disable iff (!rst_n) count==7 |-> (f_if.almostfull);

almostfull flag	
If reset isn't active, fifo is empty , raise the empty flag	@(posedge clk) disable iff (!rst_n) count==0 |-> (f_if.empty)
If there's only one place filled with data in fifo, raise the almostempty	@(posedge clk) disable iff (!rst_n) count==3'd1 |-> (f_if.almostempty)
If the fifo is full and we try to write, raise the overflow flag	@(posedge clk) disable iff (!rst_n) (f_if.full && wr_en) |=> (f_if.overflow)
If the fifo is empty and we try to read, raise the underflow flag	@(posedge clk) disable iff (!rst_n) (f_if.empty && rd_en) |=> (f_if.underflow
If write operation is successful, raise wr_ack	@(posedge clk) disable iff (!rst_n) (!f_if.full && wr_en) |=> (f_if.wr_ack)
If write operation is successful	@(posedge clk)disable iff(!rst_n) (!f_if.full&&wr_en)|=>(wr_ptr==$past(wr_ptr)+1'b1)

increment wr_ptr	
If read operation is successful increment rd_ptr	@(posedge clk)disable iff (!rst_n)(!f_if.empty&&rd_en)|=>(rd_ptr==$past(rd_ptr)+1'b1)
If wr_en & rd_en are high and fifo is empty: write only	@(posedge clk)disable iff(!rst_n) (wr_en&&rd_en&&f_if.empty)|=>((f_if.wr_ack)&&(wr_ptr==$past(wr_ptr)+1
If wr_en & rd_en are high and fifo is full: read only	@(posedge clk)disable iff(!rst_n) (wr_en&&rd_en&&f_if.full)|=>((!f_if.wr_ack)&&(rd_ptr==$past(rd_ptr)+1'b1
If write operation is successful, increment count	@(posedge clk) disable iff (!rst_n) (wr_en&&!rd_en&&!f_if.full)|=>(count==$past(count)+1'b1)
If read operation is successful, decrement count	@(posedge clk) disable iff (!rst_n) (!wr_en&&rd_en&&!f_if.empty)|=>(count==$past(count)-1'b1)