FIFO USING UVM

MAIN PROJECT

AHMED HOSSAM ALGAMAL

Introduction:

UVM FIFO Project

Create a full UVM environment for the FIFO implemented in the SV project.

Steps:

- Create a full environment for the FIFO design.
- Add constraints in the sequence item class.
- Add covergroups, coverpoints in the coverage collector class.
- Add assertions and bind it in the top module.
- Split the FIFO main sequence into multiple sequences based on your verification plan, for example:
    - write_only_sequence
    - read_only_sequence
    - write_read_sequence
- Note: You must not stick to the work done in your SV project. You are free to add or modify in the assertions, covergroups or constraints to enrich your verification.

Requirements:

- Verification plan
- Draw your UVM testbench showing the UVM structure using powerpoint, draw.io or MS Visio
    - Write a section where you will describe in details how the UVM testbench work, from the top module then driving the interface then monitoring and the analyzing the output
- Code Coverage report
- Functional Coverage report
- Sequential Domain Coverage report
- Bug report
- Sections with QuestaSim snippets to for each UVM sequence and how the interface is driven in each sequence using the waveform snippets

Submission file:

.rar file containing the following:

- PDF file having the requirements
- Testbench and design files
- Do file to run simulation (The project will not be graded if the do file is not working or missing)

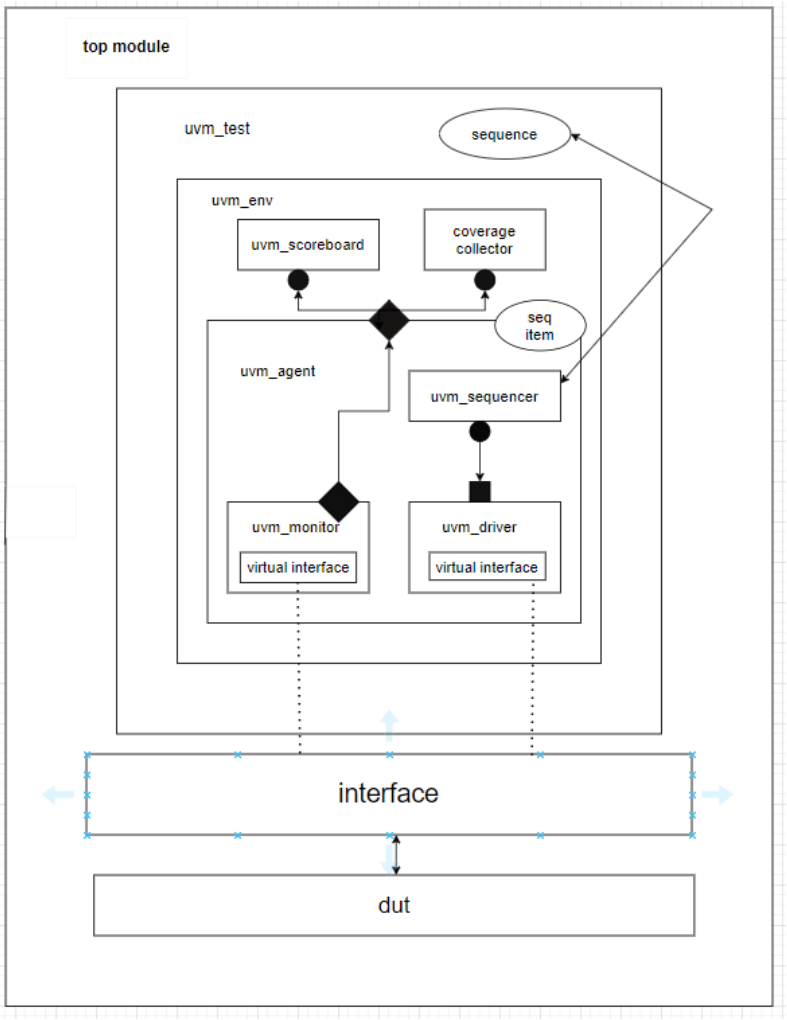
Also, provide a table in your PDF file showing the assertions used as follows

Feature	Assertion
Whenever the FIFO is full, wr_ack is always = 0	@(posedge clk) (full |-> !wr_ack)

Verification plan:

Fifo_1	When the reset is asserted, the overflow and wr_Ack must be loaded with reset values which is zero	Directed at the start of the simulation	-	Verify the values of the register after reset
Fifo_2	Verify the writing operation by raising wr_en	Randomized during simulation under constraints to write 70% of the time	Cross coverage with rd_en and all the flags	Verify the wr_ack in the assertions
Fifo_3	Verify the reading operation by raising rd_en	Randomized during simulation under constraints to write 30% of the time	Cross coverage with wr_en and all the flags	Verify the data_out in the check task in the testbench
Fifo_4	Verify the reading and writing at the same time but the count is not 8 or 0	- Randomized during simulation under constraints to write 70% of the time - Randomized during simulation under constraints to write 30% of the time	Cross coverage with all the flags	Verify the wr_ack in the assertions Verify the data_out in the check task in the testbench
Fifo_5	Verify the reading when rd_en and wr_en are high at the same time but the count is 8	- Randomized during simulation under constraints to write 70% of the time - Randomized during simulation under constraints to write 30% of the time	Cross coverage with all the flags	Verify the data_out in the check task in the testbench
Fifo_6	Verify the writing when rd_en and wr_en are high at the same time but the count is 0	- Randomized during simulation under constraints to write 70% of the time - Randomized during simulation under constraints to write 30% of the time	Cross coverage with all the flags	Verify the wr_ack in the assertions
Fifo_7	Verify the almost full flag	Randomized during simulation	Cross coverage with wr_en and rd_en	Verify the almost_full in the assertions
Fifo_8	Verify the almost empty flag	Randomized during simulation	Cross coverage with wr_en and rd_en	Verify the almost_empty in the assertions
Fifo_9	Verify the_full flag	Randomized during simulation	Cross coverage with wr_en and rd_en	Verify_full in the assertions
Fifo_10	Verify the empty flag	Randomized during simulation	Cross coverage with wr_en and rd_en	Verify_empty in the assertions
Fifo_11	Verify the overflow flag when the full flag is high and wr_en is high too	Randomized during simulation	Cross coverage with wr_en and rd_en	Verify_overflow in the assertions
Fifo_12	Verify the underflow flag when the empty flag is high and rd_en is high too	Randomized during simulation	Cross coverage with wr_en and rd_en	Verify_underflow in the assertions
Fifo_13	Verify the wr_ack flag this happen when wr_en is high and count <8	Randomized during simulation	Cross coverage with wr_en and rd_en	Verify_wr_ack in the assertions

UVM structure:



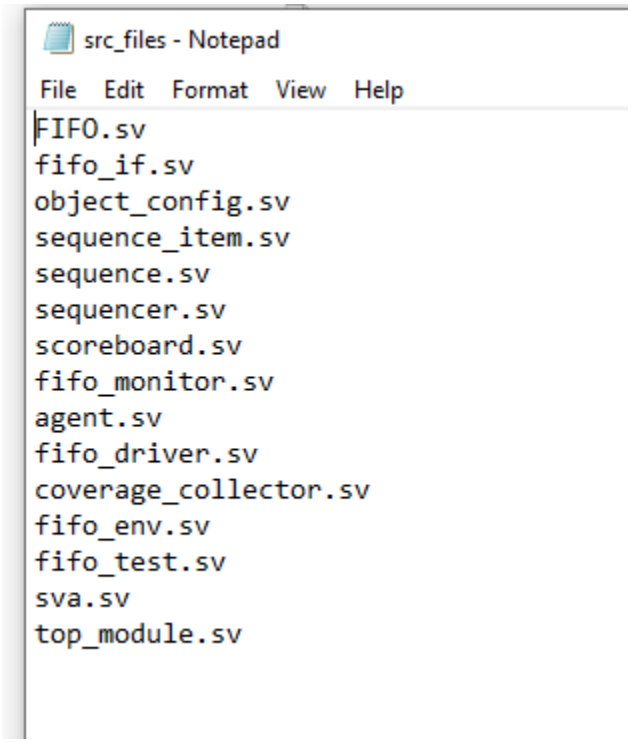
Summary about the process:

First of all the top module instantiate the interface and the DUT and generate the CLK and send the interface in the database and run uvm test and let uvm_test to have the access to get it and when the test gets it there is an object created which has pointer pointing to the interface and then the test sends the object to database another time and the test creates environment and the environment has uvm_agent which has created also uvm_monitor and uvm_driver and the agent gets the object from the database and pass the pointer to both the driver who has the responsibility to drive the stimulus and also the monitor which will take the values to display it and the agent will send also to the scoreboard to check the functionality of the design and to the coverage collector to check the coverage and all the data sent are in the form of sequence items “transaction”.

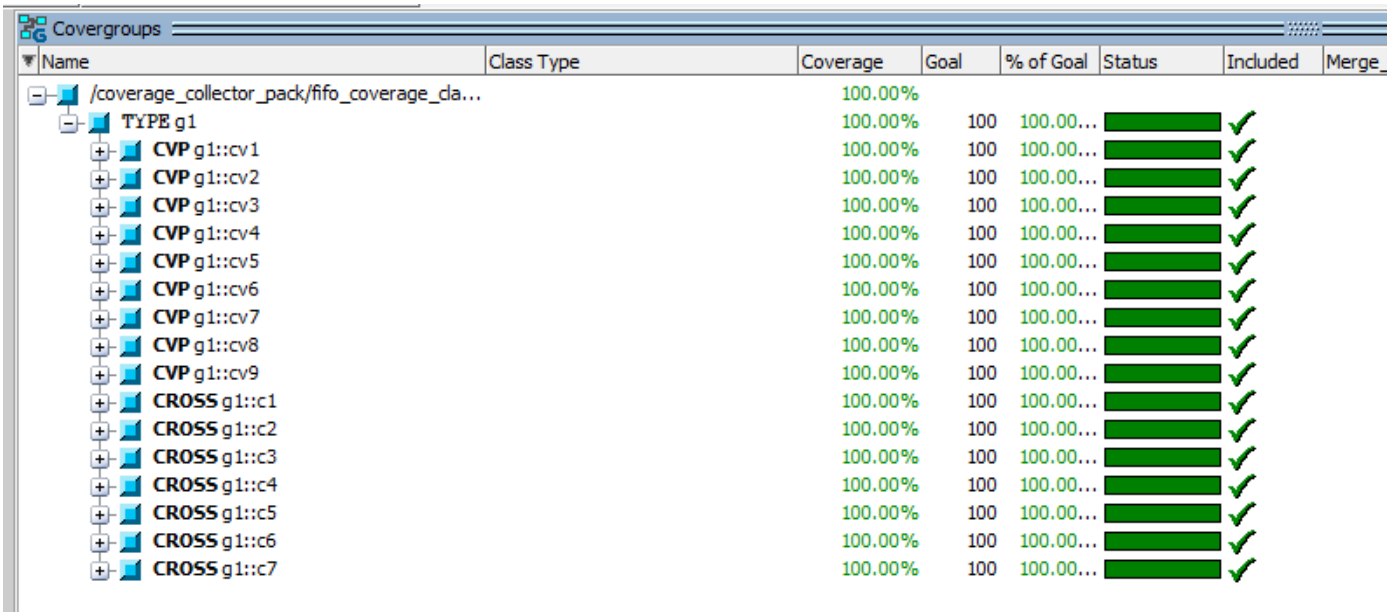
Do file:

```
vlib work
vlog -f src_files.txt +cover -covercells
vsim -voptargs=+acc work.top -classdebug -uvmcontrol=all -cover
add wave /top/fifo_interface/*
add wave -position insertpoint  \
sim:/top/dut/wr_ptr \
sim:/top/dut/rd_ptr \
sim:/top/dut/mem \
sim:/top/dut/count
coverage save top.ucdb -onexit
run -all
#quit -sim
#vcover report top.ucdb -details -all -annotate -output cover_rpt.txt
```

Src_files:



Coverage:

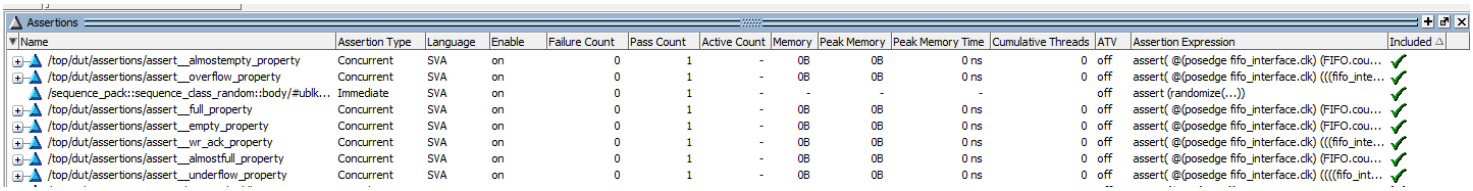


Covergroups

Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_
/coverage_collector_pack/fifo_coverage_cla...		100.00%					
TYPE g1		100.00%	100	100.00...			
CVP g1::cv1		100.00%	100	100.00...			
CVP g1::cv2		100.00%	100	100.00...			
CVP g1::cv3		100.00%	100	100.00...			
CVP g1::cv4		100.00%	100	100.00...			
CVP g1::cv5		100.00%	100	100.00...			
CVP g1::cv6		100.00%	100	100.00...			
CVP g1::cv7		100.00%	100	100.00...			
CVP g1::cv8		100.00%	100	100.00...			
CVP g1::cv9		100.00%	100	100.00...			
CROSS g1::c1		100.00%	100	100.00...			
CROSS g1::c2		100.00%	100	100.00...			
CROSS g1::c3		100.00%	100	100.00...			
CROSS g1::c4		100.00%	100	100.00...			
CROSS g1::c5		100.00%	100	100.00...			
CROSS g1::c6		100.00%	100	100.00...			
CROSS g1::c7		100.00%	100	100.00...			

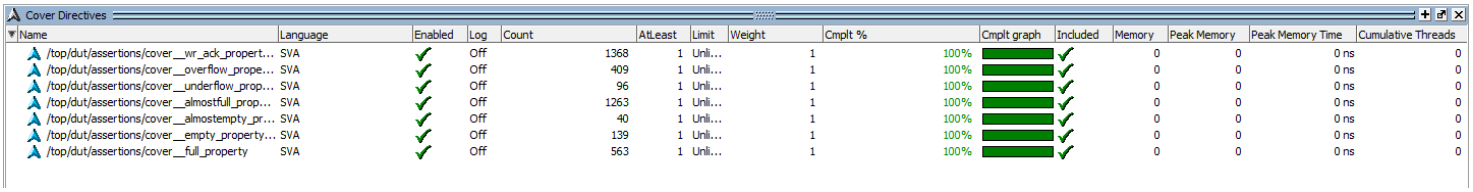
```
================================================================================
=== Instance: /top/dut
=== Design Unit: work.FIFO
================================================================================
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        27        27         0   100.00%

================================Branch Details================================

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

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

Toggle Coverage for instance /top/dut --

                                          Node      1H->0L      0L->1H  "Coverage"
                                          ----------------------------------------
                                     count[3-0]           1           1      100.00
                                    rd_ptr[2-0]           1           1      100.00
                                    wr_ptr[2-0]           1           1      100.00

Total Node Count     =         10
Toggled Node Count   =         10
Untoggled Node Count =          0

Toggle Coverage      =     100.00% (20 of 20 bins)
```

Assertions:



Assertions

Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Count	Memory	Peak Memory	Peak Memory Time	Cumulative Threads	ATV	Assertion Expression	Included
/top/dut/assertions/assert__almostempty_property	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge fifo_interface.clk) (FIFO.cou...	✓
/top/dut/assertions/assert__overflow_property	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge fifo_interface.clk) (((fifo_inte...	✓
/sequence_pack::sequence_class_random::body/#ublk...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/top/dut/assertions/assert__full_property	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge fifo_interface.clk) (FIFO.cou...	✓
/top/dut/assertions/assert__empty_property	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge fifo_interface.clk) (FIFO.cou...	✓
/top/dut/assertions/assert__wr_ack_property	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge fifo_interface.clk) (((fifo_inte...	✓
/top/dut/assertions/assert__almostfull_property	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge fifo_interface.clk) (FIFO.cou...	✓
/top/dut/assertions/assert__underflow_property	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge fifo_interface.clk) ((((fifo_int...	✓



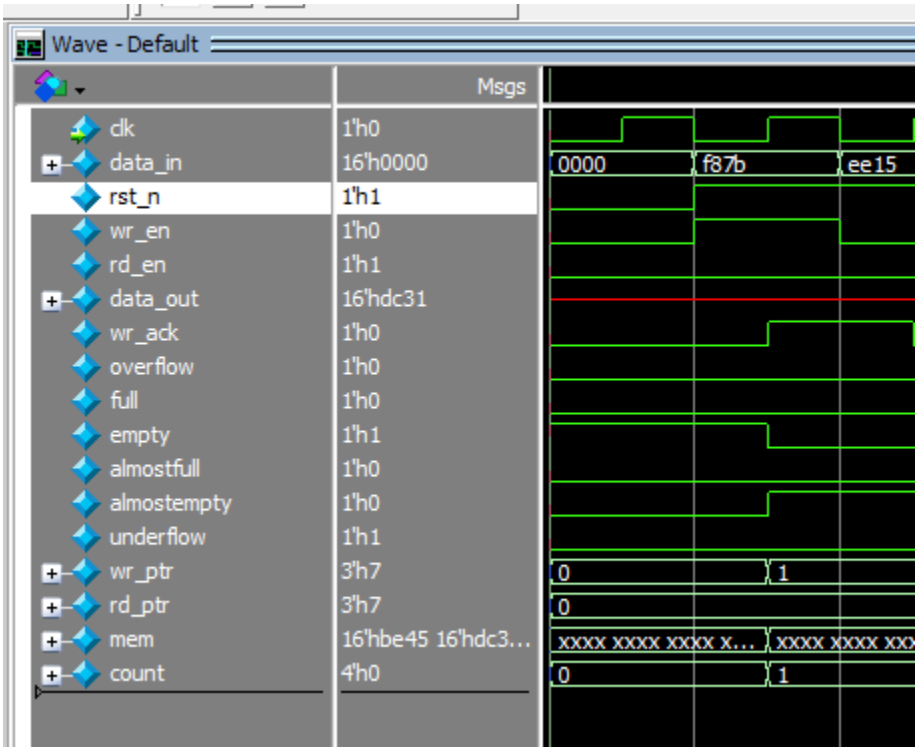
Cover Directives

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
/top/dut/assertions/cover__wr_ack_propert...	SVA	✓	Off	1368	1	Unli...	1	100%		✓	0	0	0 ns	0
/top/dut/assertions/cover__overflow_prope...	SVA	✓	Off	409	1	Unli...	1	100%		✓	0	0	0 ns	0
/top/dut/assertions/cover__underflow_prop...	SVA	✓	Off	96	1	Unli...	1	100%		✓	0	0	0 ns	0
/top/dut/assertions/cover__almostfull_prop...	SVA	✓	Off	1263	1	Unli...	1	100%		✓	0	0	0 ns	0
/top/dut/assertions/cover__almostempty_pr...	SVA	✓	Off	40	1	Unli...	1	100%		✓	0	0	0 ns	0
/top/dut/assertions/cover__empty_property...	SVA	✓	Off	139	1	Unli...	1	100%		✓	0	0	0 ns	0
/top/dut/assertions/cover__full_property	SVA	✓	Off	563	1	Unli...	1	100%		✓	0	0	0 ns	0

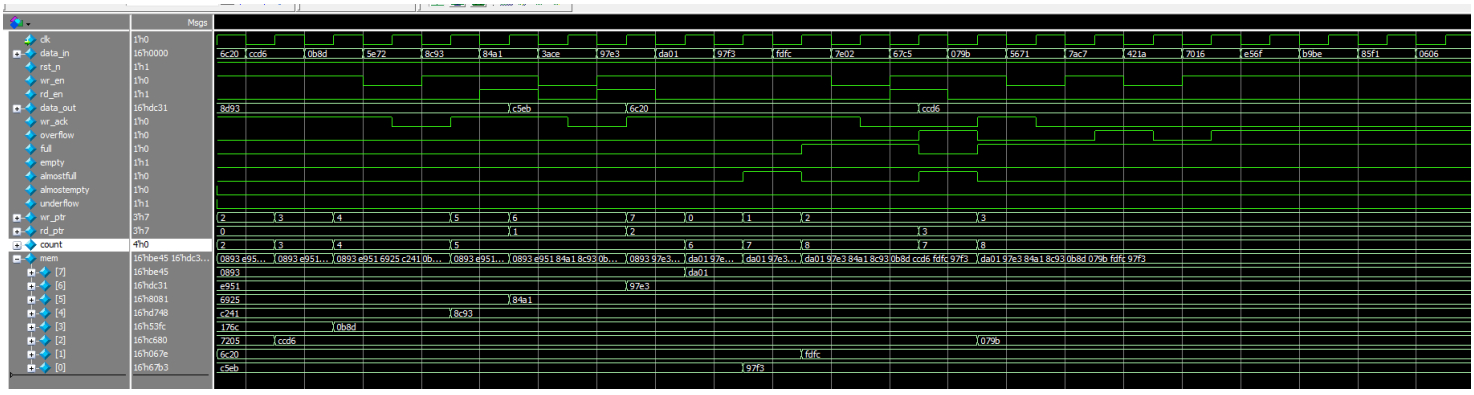
Uvm report summary:

```
# UVM_INFO scoreboard.sv(65) @ 4402: uvm_test_top.env.sb [report_phase] TOTAL SUCCESSFUL TRANSACTIONS:       2201
# UVM_INFO scoreboard.sv(66) @ 4402: uvm_test_top.env.sb [report_phase] TOTAL FAILED TRANSACTIONS:          0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   16
# 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]    10
# ** Note: $finish    : C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 4402 ns  Iteration: 61  Instance: /top
```

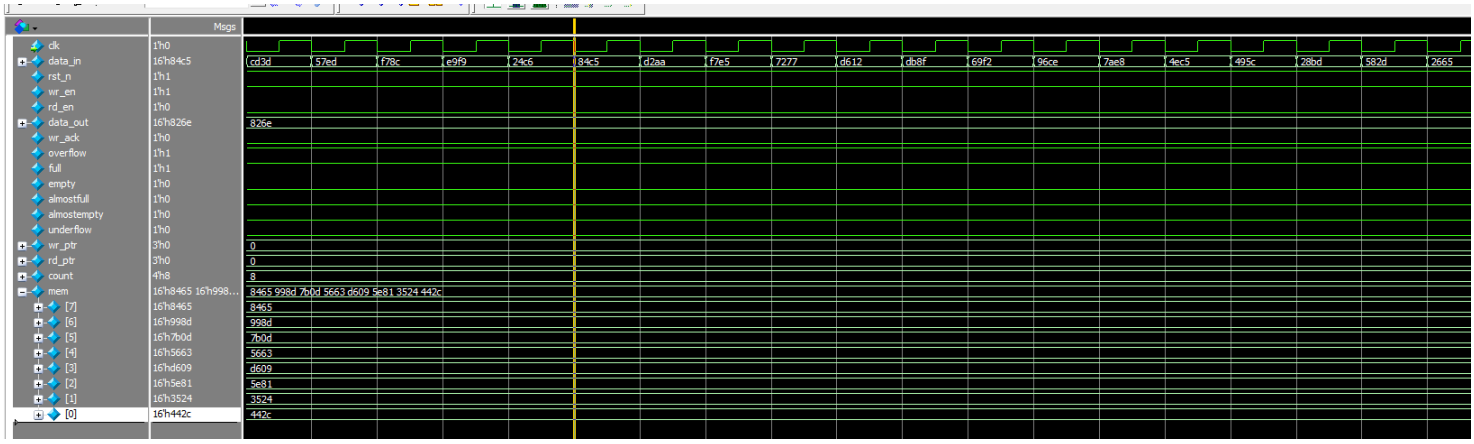
Snippets:
Reset sequence:



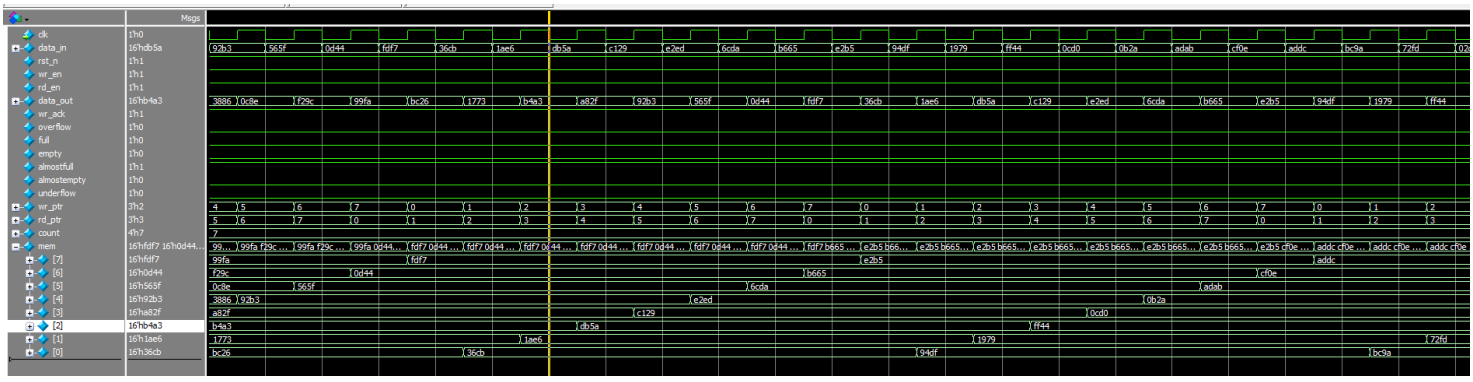
Random sequence:



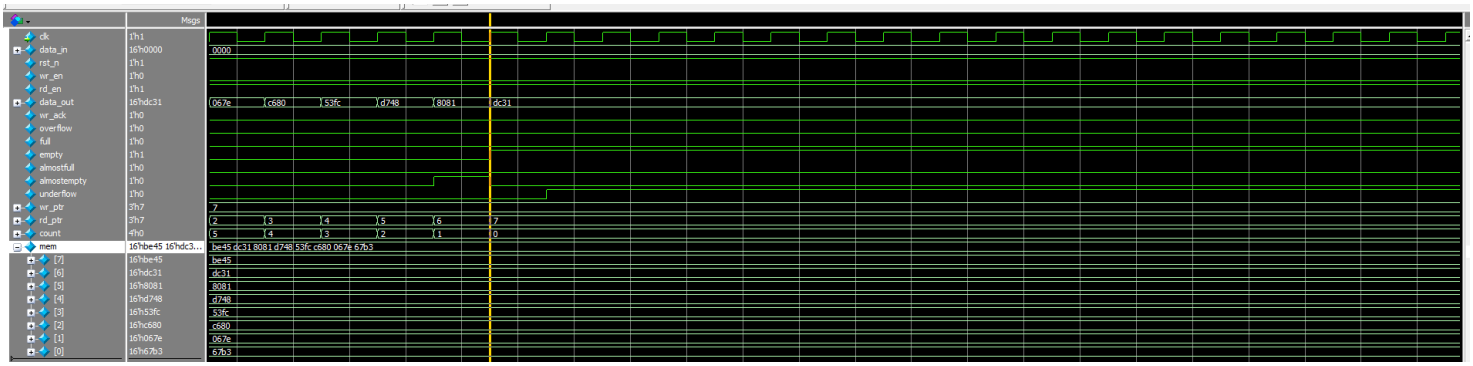
Write sequence:



Write and read sequence:



Read sequence:



Design :

```
module FIFO(fifo_if.dut fifo_interface);
localparam max_fifo_addr = $clog2(fifo_interface.FIFO_DEPTH);
reg [fifo_interface.FIFO_WIDTH-1:0] mem [fifo_interface.FIFO_DEPTH-1:0];
reg [max_fifo_addr-1:0] wr_ptr, rd_ptr;
reg [max_fifo_addr:0] count;
always @(posedge fifo_interface.clk or negedge fifo_interface.rst_n) begin /// always block in order to read
    if (!fifo_interface.rst_n) begin
        rd_ptr <= 0;
    end
    else if (fifo_interface.rd_en && count != 0) begin
        fifo_interface.data_out <= mem[rd_ptr];
        rd_ptr <= rd_ptr + 1;
    end
    else begin
     if (fifo_interface.empty && fifo_interface.rd_en ) begin
    fifo_interface.underflow<=1;                                    // edited as it was combinational and it must be sequential according to the specifications
     end
    else fifo_interface.underflow<=0;
    end
end
always @(posedge fifo_interface.clk or negedge fifo_interface.rst_n) begin      /// always block in order to write
    if (!fifo_interface.rst_n) begin
        wr_ptr <= 0;
    end
    else if (fifo_interface.wr_en && count < fifo_interface.FIFO_DEPTH ) begin
        mem[wr_ptr] <= fifo_interface.data_in;
        fifo_interface.wr_ack <= 1;
        wr_ptr <= wr_ptr + 1;
    end
```

```
    end

    end
always @(posedge fifo_interface.clk or negedge fifo_interface.rst_n) begin
    if (!fifo_interface.rst_n) begin
        count <= 0;
    end
    else begin
        if  ( ({fifo_interface.wr_en, fifo_interface.rd_en} == 2'b10) && !fifo_interface.full)
            count <= count + 1;
        else if ( ({fifo_interface.wr_en, fifo_interface.rd_en} == 2'b01) && !fifo_interface.empty)
            count <= count - 1;
            else if (({fifo_interface.wr_en, fifo_interface.rd_en} == 2'b11)  && fifo_interface.empty) /// was missing to react on wr_en and rd_en are high together and the fifo is empty to write only
            count <= count + 1;
            else if (({fifo_interface.wr_en, fifo_interface.rd_en} == 2'b11)  && fifo_interface.full)   /// was missing to react on wr_en and rd_en are high together and the fifo is full to read only
            count <= count - 1;
    end
end

always @(posedge fifo_interface.clk or negedge fifo_interface.rst_n) begin /// always block for overflow and wr_ack flags
    if (!fifo_interface.rst_n) begin
        fifo_interface.overflow <= 0;                           // added inorder to the flag down
        fifo_interface.wr_ack<=0;
        fifo_interface.underflow<=0;                              // added inorder to the flag down
    end
    else if (fifo_interface.wr_en && count==8 ) begin
        fifo_interface.wr_ack<=0;
        fifo_interface.overflow<=1;
    end
else if (fifo_interface.wr_en && !fifo_interface.full) begin
        fifo_interface.overflow<=0;
        fifo_interface.wr_ack<=1;
end
        else begin
        fifo_interface.overflow<=0;
fifo_interface.wr_ack<=0;
        end
end
assign fifo_interface.full = (count == fifo_interface.FIFO_DEPTH)? 1 : 0;
assign fifo_interface.empty = (count == 0)? 1 : 0;
assign fifo_interface.almostfull = (count == fifo_interface.FIFO_DEPTH-1)? 1 : 0;
assign fifo_interface.almostempty = (count == 1)? 1 : 0;


endmodule
```

Fifo_if:

```
interface fifo_if(clk);
input clk;
parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;
logic [FIFO_WIDTH-1:0] data_in;
logic  rst_n, wr_en, rd_en;
logic [FIFO_WIDTH-1:0] data_out;
logic wr_ack, overflow;
logic full, empty, almostfull, almostempty, underflow;
//logic [2:]

modport dut (input clk,data_in,rst_n, wr_en, rd_en, output data_out,wr_ack, overflow,full, empty, almostfull, almostempty, underflow);

endinterface //fifo_if
```

Sequence item:

```
package sequence_item_pack;
import uvm_pkg::*;
`include "uvm_macros.svh"
class seq_item_class extends uvm_sequence_item;
`uvm_object_utils(seq_item_class)
parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;
rand bit [FIFO_WIDTH-1:0] data_in;
rand bit rst_n, wr_en, rd_en;
logic [FIFO_WIDTH-1:0] data_out;
logic wr_ack, overflow;
logic full, empty, almostfull, almostempty, underflow;
integer RD_EN_ON_DIST=0,WR_EN_ON_DIST=0;

function new(string name= "seq_item_class");
    super.new(name);
    RD_EN_ON_DIST=30;
     WR_EN_ON_DIST=70;
    endfunction
    function string convert2string();
    return $sformatf ("%s reset =%b ,data in=%d ,write enable=%d , read enable=%d,data_out=%h ,wr_ack=%d,overflow=%d,full=%s,empty=%d ,
    almost full=%b,almost empty=%d,underflow=%h  ",
     super.convert2string(), rst_n,data_in,wr_en,rd_en, data_out,wr_ack,overflow,full,empty ,almostfull,almostempty,underflow ) ;
    endfunction

    function string convert2string_stim();
    return $sformatf ("%s reset =%b ,data in=%d ,write enable=%d , read enable=%d ",
     super.convert2string(), rst_n,data_in,wr_en,rd_en) ;
    endfunction


constraint trans {
rst_n dist {1:=99,0:=1};
wr_en dist {1:=WR_EN_ON_DIST,0:=100-WR_EN_ON_DIST};
rd_en dist {1:=RD_EN_ON_DIST,0:=100-RD_EN_ON_DIST};
}
endclass
endpackage
```

Sequencer:

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

Object configuration:

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

Sequence:

```
package sequence_pack;
import uvm_pkg::*;
import sequence_item_pack::*;

`include "uvm_macros.svh"
class sequence_class_reset extends uvm_sequence #(seq_item_class);
`uvm_object_utils(sequence_class_reset)
seq_item_class seq_item;
function new(string name= "sequence_class_reset");
    super.new(name);
    endfunction
    task body;
    seq_item =seq_item_class::type_id::create("seq_item");
    start_item(seq_item);
    seq_item.rst_n=0;
    finish_item(seq_item);
    endtask
endclass
class sequence_class_write_only extends uvm_sequence #(seq_item_class);
`uvm_object_utils(sequence_class_write_only)
seq_item_class seq_item;
function new(string name= "sequence_class_write_only");
    super.new(name);
    endfunction
    task body;
    repeat (100)
    begin
        seq_item =seq_item_class::type_id::create("seq_item");
        start_item(seq_item);
        seq_item.wr_en=1;
        seq_item.rd_en=0;
        seq_item.rst_n=1;
  seq_item.data_in=$random;
finish_item(seq_item);
    end
    endtask
endclass
```

```
class sequence_class_read_only extends uvm_sequence #(seq_item_class);
`uvm_object_utils(sequence_class_read_only)
seq_item_class seq_item;
function new(string name= "sequence_class_read_only");
    super.new(name);
    endfunction
    task body;
    repeat (100)
    begin
        seq_item =seq_item_class::type_id::create("seq_item");
        start_item(seq_item);
        seq_item.rd_en=1;
        seq_item.wr_en=0;
        seq_item.rst_n=1;
finish_item(seq_item);
    end
    endtask
endclass
class sequence_class_write_and_read extends uvm_sequence #(seq_item_class);
`uvm_object_utils(sequence_class_write_and_read)
seq_item_class seq_item;
function new(string name= "sequence_class_write_and_read");
    super.new(name);
    endfunction
    task body;
    repeat (1000)
    begin
        seq_item =seq_item_class::type_id::create("seq_item");
        start_item(seq_item);
        seq_item.rd_en=1;
        seq_item.wr_en=1;
        seq_item.rst_n=1;
  seq_item.data_in=$random;
finish_item(seq_item);
    end
    endtask
endclass
```

```
    class sequence_class_random extends uvm_sequence #(seq_item_class);
`uvm_object_utils(sequence_class_random)
seq_item_class seq_item;
function new(string name= "sequence_class_random");
    super.new(name);
    endfunction
    task body;
    repeat (1000)
    begin
        seq_item =seq_item_class::type_id::create("seq_item");
         start_item(seq_item);
        assert(seq_item.randomize());
finish_item(seq_item);
    end
    endtask
    endclass


endpackage
```

Driver:

```
package fifo_driver_pack;
import fifo_config_pack::*;
import sequence_item_pack::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class fifo_driver extends uvm_driver#(seq_item_class);
`uvm_component_utils(fifo_driver)
virtual fifo_if fifo_vif;
seq_item_class 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=seq_item_class::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_stim(),UVM_HIGH)
     end
     endtask

endclass //fifo_driver extends uvm_driver

endpackage
```

Monitor:

```
package fifo_monitor;
import uvm_pkg::*;
`include "uvm_macros.svh"
import sequence_item_pack::*;
class fifo_monitor extends uvm_monitor;
`uvm_component_utils(fifo_monitor)
virtual fifo_if fifo_vif;
seq_item_class rsp_seq_item;
uvm_analysis_port #(seq_item_class) 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=seq_item_class::type_id::create("rsp_seq_item");
        @(negedge fifo_vif.clk);
        rsp_seq_item.rst_n=fifo_vif.rst_n;
        rsp_seq_item.data_in=fifo_vif.data_in;
        rsp_seq_item.wr_en=fifo_vif.wr_en;
        rsp_seq_item.rd_en=fifo_vif.rd_en;
        rsp_seq_item.wr_ack=fifo_vif.wr_ack;
        rsp_seq_item.empty=fifo_vif.empty;
        rsp_seq_item.almostempty=fifo_vif.almostempty;
        rsp_seq_item.full=fifo_vif.full;
        rsp_seq_item.almostfull=fifo_vif.almostfull;
        rsp_seq_item.overflow=fifo_vif.overflow;
        rsp_seq_item.underflow=  fifo_vif.underflow;
        rsp_seq_item.data_out=fifo_vif.data_out;
        mon_ap.write(rsp_seq_item);
        `uvm_info("run_phase",rsp_seq_item.convert2string(),UVM_HIGH)
    end
    endtask
endclass //fifo_monitor extends superClass
endpackage
```

Agent :

```
1   package agent_pack;
2   import sequencer_pack::*;
3   import fifo_driver_pack::*;
4   import fifo_monitor::*;
5   import fifo_config_pack::*;
6   import sequence_item_pack::*;
7   import uvm_pkg::*;
8   `include "uvm_macros.svh"
9   class agent_class extends uvm_agent;
10  `uvm_component_utils(agent_class);
11  sequencer_class sqr;
12  fifo_driver driv;
13  fifo_monitor mon;
14  fifo_config cfg;
15  uvm_analysis_port #(seq_item_class) agt_ap;
16      function new (string name= "agent_class",uvm_component parent =null);
17  super.new(name,parent);
18  endfunction
19   function void build_phase(uvm_phase phase);
20      super.build_phase(phase);
21          if (!uvm_config_db #(fifo_config)::get(this,"","KEY",cfg))
22      `uvm_fatal("build_phase","yallahwayyyyyyyy");
23       sqr=sequencer_class::type_id::create("sqr",this);
24       driv=fifo_driver::type_id::create("driv",this);
25       mon=fifo_monitor::type_id::create("mon",this);
26       agt_ap=new("agt_ap",this);
27       endfunction
28       function void connect_phase(uvm_phase phase);
29      super.connect_phase(phase);
30       driv.fifo_vif=cfg.fifo_vif;
31       mon.fifo_vif=cfg.fifo_vif;
32      driv.seq_item_port.connect(sqr.seq_item_export);
33      mon.mon_ap.connect(agt_ap);
34      endfunction
35  endclass //agent_class extends superClass
36  endpackage
```

Coverage_collector:

```
package coverage_collector_pack;
import sequence_item_pack::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class fifo_coverage_class extends uvm_component;
`uvm_component_utils(fifo_coverage_class)
uvm_analysis_export #(seq_item_class) cov_export;
uvm_tlm_analysis_fifo #(seq_item_class) cov_fifo;
seq_item_class seq_item_cov;
covergroup g1;
        cv1: coverpoint seq_item_cov.wr_en;
        cv2: coverpoint seq_item_cov.rd_en;
        cv3: coverpoint seq_item_cov.overflow;
        cv4: coverpoint seq_item_cov.almostempty;
        cv5: coverpoint seq_item_cov.empty;
        cv6: coverpoint seq_item_cov.almostfull;
        cv7: coverpoint seq_item_cov.underflow;
        cv8: coverpoint seq_item_cov.full;
        cv9: coverpoint seq_item_cov.wr_ack;
        c1:cross cv1,cv2,cv8
        {
            ignore_bins read_full = binsof(cv2) intersect {1} && binsof(cv8) intersect {1};
        }
        c2: cross cv1,cv2,cv6;
        c3:cross cv1,cv2,cv5;
        c4:cross cv1,cv2,cv4;
        c5:cross cv1,cv2,cv3
        {
            ignore_bins wr_en_overflow = binsof(cv1) intersect{0} && binsof(cv3) intersect{1};
        }
        c6:cross cv1,cv2,cv7
        {
            ignore_bins read_underflow = binsof(cv2) intersect{0} && binsof(cv7) intersect{1};
        }
        c7:cross cv1,cv2,cv9
        {
            ignore_bins wr_en_ack = binsof(cv1) intersect{0} && binsof(cv9) intersect{1};
        }
endgroup
```

```
    function new(string name="fifo_coverage_class",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 //fifo_coverage_class extends superClass

endpackage
```

Fifo_scoreboard:

```
package scoreboard_pack;
import uvm_pkg::*;
import sequence_item_pack::*;
`include "uvm_macros.svh"
class scoreboard_class extends uvm_scoreboard;
`uvm_component_utils(scoreboard_class)
uvm_analysis_export #(seq_item_class) sb_export;
uvm_tlm_analysis_fifo#(seq_item_class) sb_fifo;
seq_item_class seq_item_sb;
logic [16-1:0] data_out_ref;
int error=0;
logic [16-1:0] gold [$];
int correct=0;
function new(string name= "scoreboard_class",uvm_component parent =null);
    super.new(name,parent);
    endfunction //new()
     function void build_phase (uvm_phase phase);
    super.build_phase(phase);
    sb_export= new("sb_export",this);
    sb_fifo=new("sb_fifo",this);
     endfunction
     function void connect_phase(uvm_phase phase);
     super.connect_phase(phase);
     sb_export.connect(sb_fifo.analysis_export);
     endfunction
     task  run_phase(uvm_phase phase);
     super.run_phase(phase);
     forever begin
        sb_fifo.get(seq_item_sb);
        ref_model(seq_item_sb);
        if (seq_item_sb.data_out!=data_out_ref ) begin
            `uvm_error("run_phase",$sformatf("comparison failed, DUT:%s while ref_out:%h",seq_item_sb.convert2string(),data_out_ref));
            error++;
        end
        else begin
            `uvm_info("run_phase",$sformatf("correct ",seq_item_sb.convert2string()),UVM_HIGH);
            correct++;
        end end
     endtask //
```

```
     task  ref_model(seq_item_class seq_item_chk);
     if (!seq_item_chk.rst_n)
  gold.delete();
  else if (seq_item_chk.rd_en &&seq_item_chk.wr_en &&gold.size()!=0 &&gold.size()!=8)
  begin
    data_out_ref=gold.pop_front();
    gold.push_back(seq_item_chk.data_in);
  end
  else if (seq_item_chk.rd_en &&seq_item_chk.wr_en &&gold.size()==0)
  gold.push_back(seq_item_chk.data_in);
  else if(seq_item_chk.rd_en &&seq_item_chk.wr_en &&gold.size()==8)
  data_out_ref=gold.pop_front();
  else if (seq_item_chk.rd_en && gold.size()!=0)
  data_out_ref=gold.pop_front();
  else if ( gold.size()<8 &&seq_item_chk.wr_en)
  gold.push_back(seq_item_chk.data_in);

     endtask //
     function void report_phase(uvm_phase phase);
     super.report_phase(phase);
     `uvm_info("report_phase",$sformatf("TOTAL SUCCESSFUL TRANSACTIONS:%d",correct),UVM_MEDIUM);
     `uvm_info("report_phase",$sformatf("TOTAL FAILED TRANSACTIONS:%d",error),UVM_MEDIUM);

     endfunction
  endclass
  endpackage
```

Fifo_env:

```
package fifo_env_pack;
import uvm_pkg::*;
import coverage_collector_pack::*;
import scoreboard_pack::*;
import agent_pack::*;
`include "uvm_macros.svh"
class fifo_env extends uvm_env;
`uvm_component_utils(fifo_env)
agent_class agt;
scoreboard_class sb;
fifo_coverage_class cov;
    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=agent_class::type_id::create("agt",this);
    sb=scoreboard_class::type_id::create("sb",this);
    cov=fifo_coverage_class::type_id::create("cov",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 //fifo_env extends superClass

endpackage
```

Fifo_test:

```
package fifo_test_pack;
import fifo_env_pack::*;
import fifo_config_pack::*;
import sequence_pack::*;
import uvm_pkg::*;
`include "uvm_macros.svh"
class fifo_test extends uvm_test;
`uvm_component_utils(fifo_test)
virtual fifo_if fifo_vif;
fifo_config fifo_config_obj_test;
sequence_class_reset reset_seq;
sequence_class_write_and_read writeandread_seq;
sequence_class_write_only write_seq;
sequence_class_read_only read_seq;
sequence_class_random random_;
fifo_env env;
    function new(string name= "fifo_test",uvm_component parent =null);
    super.new(name,parent);
    endfunction //new()
    function void build_phase (uvm_phase phase);
    super.build_phase(phase);
    env=fifo_env::type_id::create("env",this);
    fifo_config_obj_test=fifo_config::type_id::create("fifo_config_obj_test");
    writeandread_seq=sequence_class_write_and_read::type_id::create("writeandread_seq",this);
    reset_seq=sequence_class_reset::type_id::create("reset_seq",this);
    write_seq=sequence_class_write_only::type_id::create("write_seq",this);
    read_seq=sequence_class_read_only::type_id::create("read_seq",this);
random_=sequence_class_random::type_id::create("random_",this);
```

```
    if (!uvm_config_db #(virtual fifo_if)::get(this,"","fifo",fifo_config_obj_test.fifo_vif))
    `uvm_fatal("build_phase","TEST - unable to get the virtual interface from the data base");
    uvm_config_db #(fifo_config)::set(this,"*","KEY",fifo_config_obj_test);
    endfunction
    task run_phase(uvm_phase phase);
    super.run_phase(phase);
    phase.raise_objection(this);
    `uvm_info("run_phase","reset_asserted",UVM_MEDIUM)
    reset_seq.start(env.agt.sqr);
     `uvm_info("run_phase","reset_deasserted",UVM_MEDIUM)
     `uvm_info("run_phase","random_asserted",UVM_MEDIUM)
     random_.start(env.agt.sqr);
     `uvm_info("run_phase","random_deasserted",UVM_MEDIUM)
    `uvm_info("run_phase","inside the write test",UVM_MEDIUM)
    write_seq.start(env.agt.sqr);
    `uvm_info("run_phase","finished the write test",UVM_MEDIUM)
    `uvm_info("run_phase","inside the write and read test",UVM_MEDIUM)
    writeandread_seq.start(env.agt.sqr);
    `uvm_info("run_phase","finished the write and read test",UVM_MEDIUM)
    `uvm_info("run_phase","inside the read test",UVM_MEDIUM)
    read_seq.start(env.agt.sqr);
    phase.drop_objection(this);
`uvm_info("run_phase","finished the read test",UVM_MEDIUM)
    endtask
endclass //fifo_test extends superClass

endpackage
```

SVA:

```
module sva (fifo_if.dut fifo_interface);
property full_property;
@(posedge fifo_interface.clk) FIFO.count==fifo_interface.FIFO_DEPTH |-> fifo_interface.full==1;
endproperty
property empty_property;
@(posedge fifo_interface.clk) FIFO.count==0 |-> fifo_interface.empty==1;
endproperty
property almostempty_property;
@(posedge fifo_interface.clk) FIFO.count==1 |-> fifo_interface.almostempty==1;
endproperty
property almostfull_property;
@(posedge fifo_interface.clk) FIFO.count==fifo_interface.FIFO_DEPTH-1 |-> fifo_interface.almostfull==1;
endproperty
property underflow_property;
@(posedge fifo_interface.clk) fifo_interface.empty && fifo_interface.rd_en &&(fifo_interface.wr_en==0) &&(fifo_interface.rst_n) |=>fifo_interface.underflow==1;
endproperty
property overflow_property;
@(posedge fifo_interface.clk) fifo_interface.wr_en && fifo_interface.full &&(fifo_interface.rst_n)|=> (fifo_interface.rst_n) |-> fifo_interface.overflow==1;
endproperty
property wr_ack_property;
@(posedge fifo_interface.clk) (fifo_interface.wr_en && fifo_interface.full==0)&&(fifo_interface.rst_n) |=> (fifo_interface.rst_n)|->fifo_interface.wr_ack==1;
endproperty
property wr_pointer;
@(posedge fifo_interface.clk) disable iff(!(fifo_interface.rst_n)) (fifo_interface.wr_en && FIFO.count < fifo_interface.FIFO_DEPTH) |=> (fifo_interface.rst_n)
|-> FIFO.wr_ptr==($past(FIFO.wr_ptr)+1)%8;
endproperty
property rd_pointer;
@(posedge fifo_interface.clk) disable iff(!(fifo_interface.rst_n)) fifo_interface.rd_en && FIFO.count != 0 |=> (fifo_interface.rst_n)
|-> FIFO.rd_ptr==($past(FIFO.rd_ptr)+1)%8;
endproperty

```

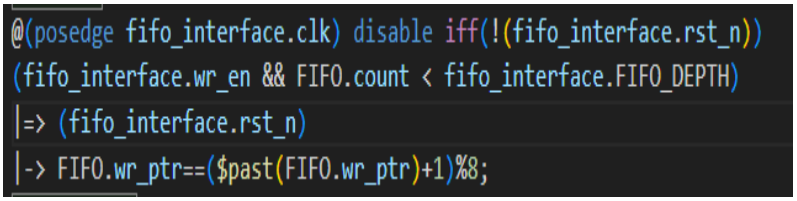
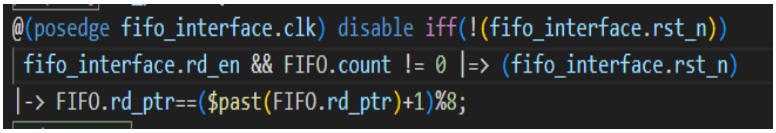
```
32  assert property (full_property);
33  cover property (full_property);
34  assert property (empty_property);
35  cover property (empty_property);
36  assert property (almostempty_property);
37  cover property (almostempty_property);
38  assert property (almostfull_property);
39  cover property (almostfull_property);
40  assert property (underflow_property);
41  cover property (underflow_property);
42  assert property (overflow_property);
43  cover property (overflow_property);
44  assert property (wr_ack_property);
45  cover property (wr_ack_property);
46
47  endmodule
```

Top module:

```
1   import uvm_pkg::*;
2   import fifo_test_pack::*;
3   `include "uvm_macros.svh"
4   module top ();
5   bit clk;
6   initial begin
7       clk=0;
8       forever begin
9           #1 clk=~clk;
10      end
11  end
12  fifo_if fifo_interface (clk);
13  FIFO dut (fifo_interface);
14  bind FIFO sva assertions (fifo_interface);
15  initial begin
16      uvm_config_db #(virtual fifo_if)::set(null,"uvm_test_top","fifo",fifo_interface);
17      run_test("fifo_test");
18  end
19  endmodule
```

Table of assertions:

feature	assertion
When count reaches 8 the full flag gets high	`@(posedge fifo_interface.clk) FIFO.count==fifo_interface.FIFO_DEPTH |-> fifo_interface.full==1;`
When count equals 0 the empty flag gets high	`@(posedge fifo_interface.clk) FIFO.count==0 |-> fifo_interface.empty==1;`
When count equals one almost empty flag gets high	`@(posedge fifo_interface.clk) FIFO.count==1 |-> fifo_interface.almostempty==1;`
When count equals 7 the almost full flag gets high	`@(posedge fifo_interface.clk) FIFO.count==fifo_interface.FIFO_DEPTH-1 |-> fifo_interface.almostfull==1;`
When count equals zero and the read enable is high the underflow flag gets high	`@(posedge fifo_interface.clk) fifo_interface.empty && fifo_interface.rd_en && (fifo_interface.rst_n) |=>fifo_interface.underflow==1;`
When count equals 8 and write enable is high the overflow flag gets high	`@(posedge fifo_interface.clk) fifo_interface.wr_en && fifo_interface.full && (fifo_interface.rst_n) |=> (fifo_interface.rst_n) |-> fifo_interface.overflow==1;`
When the full flag is low and the write enable is high then the wr_ack will be high	`@(posedge fifo_interface.clk) (fifo_interface.wr_en && fifo_interface.full==0)&&(fifo_interface.rst_n) |=> (fifo_interface.rst_n) |->fifo_interface.wr_ack==1;`

When the write enable is high and the count is <8 then the write pointer will be incremented and when it reaches 8 it returns to 0 again	
When the read enable is high and the count is not 0 then the read pointer will be incremented and when it reaches 8 it returns to 0 again	

Thank you !