## Top module :

```
import uvm_pkg :: *;
import FIFO_test_pkg::*;
import FIFO_env_pkg::*;
import FIFO_driver_pkg :: *;
import FIFO_config_pkg :: * ;
`include "uvm_macros.svh"
`include "FIFO.sv"

module top ();

bit clk ;
initial begin
forever begin
#1 clk =~ clk ;
end
end

FIFO_interface FIFO_if (clk);
FIFO dut (FIFO_if.DUT);
//bind FIFO_test FIFO_assertions assert_inst (FIFO_if) ;//

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

## FIFO_interface :

```
interface FIFO_interface (clk);

parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;
localparam max_fifo_addr = $clog2(FIFO_DEPTH);//put the local param before the signals
input bit clk;
logic [FIFO_WIDTH-1:0] data_in;
logic rst_n, wr_en, rd_en;
logic [FIFO_WIDTH-1:0] data_out;
logic wr_ack, overflow;
logic full, empty, almostfull, almostempty, underflow;
```

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

endinterface //FIFO_interface (clk)
```

## FIFO_design code with fixed bugs :

```
////////////////////////////////////////////////////////////////////////////////
// Author: Kareem Waseem
// Course: Digital Verification using SV & UVM
//
// Description: FIFO Design
//
////////////////////////////////////////////////////////////////////////////////

module FIFO (FIFO_interface.DUT FIFO_if);
  // Memory declaration
  reg [FIFO_if.FIFO_WIDTH-1:0] mem [FIFO_if.FIFO_DEPTH-1:0];
  reg [FIFO_if.max_fifo_addr-1:0] wr_ptr, rd_ptr;
  reg [FIFO_if.max_fifo_addr:0] count;

  // Write Logic
  always @(posedge FIFO_if.clk or negedge FIFO_if.rst_n) begin
    if (!FIFO_if.rst_n) begin
      wr_ptr <= 0;
      FIFO_if.wr_ack <= 0; // Reset ack signal
    end else begin
      if (FIFO_if.wr_en && !FIFO_if.full) begin
        mem[wr_ptr] <= FIFO_if.data_in;
        FIFO_if.wr_ack <= 1;
        wr_ptr <= wr_ptr + 1;
      end else begin
        FIFO_if.wr_ack <= 0;
        if (FIFO_if.full && FIFO_if.wr_en)
          FIFO_if.overflow <= 1;
        else
          FIFO_if.overflow <= 0;
      end
    end
  end

  // Read Logic
  always @(posedge FIFO_if.clk or negedge FIFO_if.rst_n) begin
```

```
    if (!FIFO_if.rst_n) begin
      rd_ptr <= 0;
    end else begin
      if (FIFO_if.rd_en && !FIFO_if.empty) begin
        FIFO_if.data_out <= mem[rd_ptr];
        rd_ptr <= rd_ptr + 1;
      end
    end
  end

  // Count Logic
  always @(posedge FIFO_if.clk or negedge FIFO_if.rst_n) begin
    if (!FIFO_if.rst_n) begin
      count <= 0;
    end else begin
      if (FIFO_if.wr_en && !FIFO_if.full && !FIFO_if.rd_en)
        count <= count + 1; // Increment count on write
      else if (FIFO_if.rd_en && !FIFO_if.empty && !FIFO_if.wr_en)
        count <= count - 1; // Decrement count on read
    end
  end

  // Status Flags Logic (Combinational)
  always @(*) begin
    FIFO_if.full = (count == FIFO_if.FIFO_DEPTH);
    FIFO_if.empty = (count == 0);
    FIFO_if.underflow = (FIFO_if.empty && FIFO_if.rd_en);
    FIFO_if.almostfull = (count == FIFO_if.FIFO_DEPTH - 1);
    FIFO_if.almostempty = (count == 1);
    // Overflow condition is already handled in the write logic.
  end

assert property (@(posedge FIFO_if.clk) (count == FIFO_if.FIFO_DEPTH) |-> FIFO_if.full
);
assert property (@(posedge FIFO_if.clk) (count == 0) |-> FIFO_if.empty );
assert property (@(posedge FIFO_if.clk) (count == FIFO_if.FIFO_DEPTH-1) |->
FIFO_if.almostfull );
assert property (@(posedge FIFO_if.clk) (count == 1) |-> FIFO_if.almostempty );
assert property (@(posedge FIFO_if.clk) (FIFO_if.empty && FIFO_if.rd_en) |->
FIFO_if.underflow);
assert property (@(posedge FIFO_if.clk) (FIFO_if.full && FIFO_if.wr_en) |=>
FIFO_if.overflow);

cover property (@(posedge FIFO_if.clk) (count == FIFO_if.FIFO_DEPTH) |->
FIFO_if.full );
cover property (@(posedge FIFO_if.clk) (count == 0) |-> FIFO_if.empty );
cover property (@(posedge FIFO_if.clk) (count == FIFO_if.FIFO_DEPTH-1) |->
FIFO_if.almostfull );
cover property (@(posedge FIFO_if.clk) (count == 1) |-> FIFO_if.almostempty );
cover property (@(posedge FIFO_if.clk) (FIFO_if.empty && FIFO_if.rd_en) |->
FIFO_if.underflow);
cover property (@(posedge FIFO_if.clk) (FIFO_if.full && FIFO_if.wr_en) |=>
FIFO_if.overflow);

endmodule
```

## FIFO_test :

```
package FIFO_test_pkg;
import uvm_pkg::*;
import FIFO_driver_pkg::*;
import FIFO_monitor_pkg::*;
import FIFO_agent_pkg::*;
import FIFO_env_pkg::*;
import FIFO_sequence_item_pkg ::*;
import FIFO_reset_sequence_pkg ::*;
import FIFO_write_only_seq_pkg ::*;
import FIFO_read_only_seq_pkg ::*;
import FIFO_read_write_seq_pkg ::*;
import FIFO_config_pkg::*;
`include "uvm_macros.svh"

class FIFO_test extends uvm_test;
`uvm_component_utils(FIFO_test)
FIFO_env env ;
FIFO_config_obj FIFO_cfg;
virtual FIFO_interface FIFO_if ;
FIFO_reset_sequence rst_seq ;
FIFO_write_only_seq write_seq ;
FIFO_read_only_seq read_seq ;
FIFO_read_write_seq rd_wrt_seq ;

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_cfg = FIFO_config_obj :: type_id::create("FIFO_cfg",this);
rst_seq = FIFO_reset_sequence:: type_id::create("rst_seq ",this);
write_seq = FIFO_write_only_seq :: type_id::create("write_seq ",this);
read_seq = FIFO_read_only_seq :: type_id::create("read_seq ",this);
```

```
rd_wrt_seq = FIFO_read_write_seq :: type_id::create("rd_wrt_seq",this);

if (!uvm_config_db #( virtual FIFO_interface) :: get
(this,"","FIFO_if",FIFO_cfg.FIFO_if)) begin
`uvm_fatal ("build_phase","unable to get the virtual interface of FIFO from the
config_db");
end
uvm_config_db #(FIFO_config_obj) :: set (this,"*","CFG",FIFO_cfg);

endfunction

task run_phase ( uvm_phase phase);
super.run_phase(phase);
phase.raise_objection(this);
rst_seq.start(env.agent.sqr);
write_seq.start(env.agent.sqr);
read_seq.start(env.agent.sqr);
rd_wrt_seq.start(env.agent.sqr);
phase.drop_objection(this);
endtask

endclass //FIFO_test extends uvm_test
endpackage
```

## FIFO_reset_sequence:

```
package FIFO_reset_sequence_pkg ;
import uvm_pkg :: * ;
import FIFO_sequence_item_pkg ::*;
`include "uvm_macros.svh"

class FIFO_reset_sequence extends uvm_sequence #(FIFO_sequence_item);
`uvm_object_utils(FIFO_reset_sequence)
FIFO_sequence_item FIFO_item ;

function new(string name = "FIFO_reset_sequence ");
super.new(name);
endfunction //new()

task body;
FIFO_item = FIFO_sequence_item :: type_id ::create ("FIFO_item");
start_item(FIFO_item);
FIFO_item.rst_n = 0 ;
FIFO_item.data_in = 0 ;
FIFO_item.wr_en = 0 ;
FIFO_item.rd_en = 0 ;
FIFO_item.full = 0 ;
FIFO_item.empty = 1 ;
FIFO_item.almostfull = 0 ;
FIFO_item.almostempty = 0 ;
FIFO_item.underflow = 0 ;
FIFO_item.overflow = 0 ;
FIFO_item.wr_ack = 0 ;
finish_item(FIFO_item);
endtask

endclass //FIFO_reset_sequence extends superClass
endpackage
```

## FIFO_write_only_sequence:

```
package FIFO_write_only_seq_pkg ;
import uvm_pkg :: * ;
import FIFO_sequence_item_pkg ::*;
`include "uvm_macros.svh"

class FIFO_write_only_seq extends uvm_sequence #(FIFO_sequence_item);
`uvm_object_utils(FIFO_write_only_seq)
FIFO_sequence_item FIFO_item ;



function new (string name = "FIFO write_only_seq");
super.new(name);
endfunction //new()


task body;
repeat (10000) begin
FIFO_item = FIFO_sequence_item :: type_id ::create ("FIFO_item");
start_item(FIFO_item);
FIFO_item.wr_en.rand_mode(0) ;
FIFO_item.rd_en.rand_mode(0) ;
FIFO_item.wr_en = 1 ;
FIFO_item.rd_en = 0 ;
assert( FIFO_item.randomize()) ;
finish_item(FIFO_item);
end
endtask
```

```
endclass //FIFO_main_sequence extends superClass
endpackage
```

## FIFO_read_only_sequence:

```
package FIFO_read_only_seq_pkg ;
import uvm_pkg :: * ;
import FIFO_sequence_item_pkg ::*;
`include "uvm_macros.svh"

class FIFO_read_only_seq extends uvm_sequence #(FIFO_sequence_item);
`uvm_object_utils(FIFO_read_only_seq)
FIFO_sequence_item FIFO_item ;

function new (string name = "FIFO read_only_seq");
super.new(name);
endfunction //new()

task body;
repeat (10000) begin
FIFO_item = FIFO_sequence_item :: type_id ::create ("FIFO_item");
start_item(FIFO_item);
FIFO_item.wr_en.rand_mode(0) ;
FIFO_item.rd_en.rand_mode(0) ;
FIFO_item.wr_en = 0 ;
FIFO_item.rd_en = 1 ;
assert(FIFO_item.randomize());
finish_item(FIFO_item);
end
endtask

endclass //FIFO_main_sequence extends superClass
endpackage
```

## FIFO_read_write_sequence:

```
package FIFO_read_write_seq_pkg ;
import uvm_pkg :: * ;
import FIFO_sequence_item_pkg ::*;
`include "uvm_macros.svh"

class FIFO_read_write_seq extends uvm_sequence #(FIFO_sequence_item);
`uvm_object_utils(FIFO_read_write_seq)
FIFO_sequence_item FIFO_item ;
```

```
function new (string name = "FIFO_read_write_seq");
super.new(name);
endfunction //new()

task body;
repeat (10000) begin
FIFO_item = FIFO_sequence_item :: type_id ::create ("FIFO_item");
start_item(FIFO_item);
assert(FIFO_item.randomize());
finish_item(FIFO_item);
end
endtask

endclass //FIFO_main_sequence extends superClass
endpackage
```

## FIFO_env :

```
package FIFO_env_pkg;
import uvm_pkg :: * ;
import FIFO_driver_pkg :: * ;
import FIFO_config_pkg :: * ;
import FIFO_scoreboard_pkg :: * ;
import FIFO_coverage_pkg :: * ;
import FIFO_agent_pkg :: * ;
`include "uvm_macros.svh"

class FIFO_env extends uvm_env;
`uvm_component_utils(FIFO_env)
FIFO_agent agent ;
FIFO_scoreboard scoreboard ;
FIFO_cvrg coverage ;
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);
agent = FIFO_agent :: type_id::create ("agent",this);
scoreboard = FIFO_scoreboard :: type_id ::create ("scoreboard",this);
coverage = FIFO_cvrg :: type_id::create ("coverage",this);
endfunction
function void connect_phase (uvm_phase phase);
agent.agt_ap.connect(scoreboard.sb_export);
agent.agt_ap.connect(coverage.cov_export);
```

```
endfunction
endclass //FIFO_env extends uvm_env
endpackage
```

## FIFO_scoreboard :

```
package FIFO_scoreboard_pkg ;
import uvm_pkg :: * ;
`include "uvm_macros.svh"
import FIFO_driver_pkg ::* ;
import FIFO_config_pkg :: *;
import FIFO_sequencer_pkg::*;
import FIFO_monitor_pkg ::*;
import FIFO_sequence_item_pkg::*;
parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;

class FIFO_scoreboard extends uvm_scoreboard;
`uvm_component_utils(FIFO_scoreboard)

uvm_analysis_export #(FIFO_sequence_item ) sb_export ;
uvm_tlm_analysis_fifo #(FIFO_sequence_item ) sb_fifo ;
FIFO_sequence_item seq_item_sb ;

localparam max_fifo_addr = $clog2(FIFO_DEPTH);//put the local param before the signals
logic [FIFO_WIDTH-1:0] out_ref ;
logic [FIFO_WIDTH-1:0] data_out ;
int error_count = 0;
int correct_count = 0;

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

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 != out_ref ) begin
`uvm_error ("run_phase","error");
error_count ++ ;
end else begin
`uvm_info ("run_phase" , "correct", UVM_MEDIUM );
correct_count ++;
end
end
endtask

task ref_model (FIFO_sequence_item out_calculated);

// Static variables to simulate FIFO memory and pointers
static bit [FIFO_WIDTH-1:0] fifo_memory[FIFO_DEPTH-1:0]; // FIFO storage
static int write_pointer = 0; // Pointer for writing data
static int read_pointer = 0; // Pointer for reading data
out_calculated = new () ;
// Initialize out_ref based on FIFO state
if (out_calculated.wr_en && !((write_pointer + 1) %
out_calculated.FIFO_DEPTH == read_pointer)) begin
// If write enable is high and FIFO is not full, write data
fifo_memory[write_pointer] = out_calculated.data_in ; // Write data into memory
write_pointer = (write_pointer + 1) % out_calculated.FIFO_DEPTH; // Move write pointer
forward
end
if (out_calculated.rd_en && (read_pointer != write_pointer)) begin
// If read enable is high and FIFO is not empty, read data
out_ref = fifo_memory[read_pointer]; // Set expected output data from read pointer
read_pointer = (read_pointer + 1) % out_calculated.FIFO_DEPTH ; // Move read pointer
forward
end else if (!out_calculated.rd_en ) begin
// If no read operation, maintain current expected output
out_ref = fifo_memory[read_pointer]; // Keep the same expected output if no read
occurs
end

endtask

function void report_phase (uvm_phase phase);
super.report_phase(phase) ;
`uvm_info ("report_phase" , $sformatf("number of correct = %0d",correct_count),
UVM_MEDIUM );
```

```
`uvm_info ("report_phase" , $sformatf("number of errors = %0d",error_count),
UVM_MEDIUM );
endfunction
endclass

endpackage
```

## FIFO_coverage :

```
package FIFO_coverage_pkg;
import uvm_pkg :: * ;
`include "uvm_macros.svh"
import FIFO_driver_pkg ::* ;
import FIFO_config_pkg :: *;
import FIFO_sequencer_pkg::*;
import FIFO_monitor_pkg ::*;
import FIFO_sequence_item_pkg::*;

class FIFO_cvrg extends uvm_component;
`uvm_component_utils(FIFO_cvrg)
uvm_analysis_export #(FIFO_sequence_item ) cov_export ;
uvm_tlm_analysis_fifo #(FIFO_sequence_item ) cov_fifo ;
FIFO_sequence_item seq_item_cov ;

covergroup FIFO_cover;
write : coverpoint seq_item_cov.wr_en ;
read : coverpoint seq_item_cov.rd_en ;
full : coverpoint seq_item_cov.full ;
almostfull : coverpoint seq_item_cov.almostfull ;
empty : coverpoint seq_item_cov.empty ;
almostempty : coverpoint seq_item_cov.almostempty ;
overflow : coverpoint seq_item_cov.overflow ;
underflow : coverpoint seq_item_cov.underflow ;
wr_ack : coverpoint seq_item_cov.wr_ack ;
a0: cross write , full ;
a1: cross write , almostfull ;
a2: cross write , wr_ack ;
a4: cross write , overflow ;
b0: cross read , empty ;
b1: cross read , almostempty ;
b2: cross read , underflow ;
endgroup

function new ( string name = "FIFO_cvrg" , uvm_component parent = null );
super.new (name,parent);
```

```
FIFO_cover = new();
endfunction

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);
FIFO_cover.sample();
end
endtask

endclass //FIFO_cvrg extends superClass
endpackage
```

## FIFO_agent :

```
package FIFO_agent_pkg;
import uvm_pkg :: * ;
import FIFO_driver_pkg :: * ;
import FIFO_config_pkg :: * ;
import FIFO_sequencer_pkg :: * ;
import FIFO_monitor_pkg :: * ;
import FIFO_sequence_item_pkg ::*;
`include "uvm_macros.svh"

class FIFO_agent extends uvm_agent;
`uvm_component_utils(FIFO_agent);
FIFO_sequencer sqr ;
FIFO_driver drv ;
FIFO_monitor mon ;
FIFO_config_obj cfg ;
uvm_analysis_port #(FIFO_sequence_item) agt_ap ;

function new(string name = "FIFO_agent " , uvm_component parent = null);
```

```
super.new(name , parent);
endfunction //new()

function void build_phase ( uvm_phase phase );
super.build_phase(phase);
if (! uvm_config_db #(FIFO_config_obj) :: get (this,"","CFG", cfg )) begin
`uvm_fatal ("build_phase","unable to get the virtual interface of FIFO from the 
config_db");
end
sqr = FIFO_sequencer :: 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_if = cfg.FIFO_if ;
mon.FIFO_if = cfg.FIFO_if ;
drv.seq_item_port.connect(sqr.seq_item_export);
mon.mon_ap.connect (agt_ap);
endfunction

endclass //FIFO_agent extends uvm_agent

endpackage
```

## FIFO_driver :

```
package FIFO_driver_pkg;
import uvm_pkg :: * ;
import FIFO_config_pkg :: * ;
import FIFO_sequence_item_pkg ::*;
`include "uvm_macros.svh"

class FIFO_driver extends uvm_driver #(FIFO_sequence_item);
`uvm_component_utils(FIFO_driver)
virtual FIFO_interface FIFO_if;
FIFO_sequence_item FIFO_item_stim ;

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
FIFO_item_stim = FIFO_sequence_item :: type_id ::create ("FIFO_item_stim");
seq_item_port.get_next_item(FIFO_item_stim);
FIFO_item_stim.clk = FIFO_if.clk;

FIFO_if.data_in = FIFO_item_stim.data_in ;
FIFO_if.wr_en = FIFO_item_stim.wr_en ;
FIFO_if.rd_en = FIFO_item_stim.rd_en ;
FIFO_if.rst_n = FIFO_item_stim.rst_n ;


@(negedge FIFO_if.clk);
seq_item_port.item_done();
`uvm_info ("run_phase", FIFO_item_stim.convert2string_stimulus(),UVM_MEDIUM);
end
endtask

endclass //className extends superClass
endpackage
```

## FIFO_monitor :

```
package FIFO_monitor_pkg ;
import uvm_pkg :: * ;
import FIFO_sequence_item_pkg ::*;
`include "uvm_macros.svh"

class FIFO_monitor extends uvm_monitor;
`uvm_component_utils(FIFO_monitor);
virtual FIFO_interface FIFO_if ;
FIFO_sequence_item FIFO_item_rsp ;
uvm_analysis_port #(FIFO_sequence_item) 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
```

```
FIFO_item_rsp = FIFO_sequence_item :: type_id ::create ("FIFO_item_rsp");
@(negedge FIFO_if.clk );

FIFO_item_rsp.data_in = FIFO_if.data_in;
FIFO_item_rsp.wr_en = FIFO_if.wr_en ;
FIFO_item_rsp.rd_en  = FIFO_if.rd_en;
FIFO_item_rsp.rst_n  = FIFO_if.rst_n  ;

FIFO_item_rsp.full = FIFO_if.full;
FIFO_item_rsp.empty = FIFO_if.empty;
FIFO_item_rsp.almostfull = FIFO_if.almostfull;
FIFO_item_rsp.almostempty = FIFO_if.almostempty;
FIFO_item_rsp.underflow = FIFO_if.underflow;
FIFO_item_rsp.overflow = FIFO_if.overflow;
FIFO_item_rsp.wr_ack = FIFO_if.wr_ack;
FIFO_item_rsp.data_out = FIFO_if.data_out;

mon_ap.write(FIFO_item_rsp);
`uvm_info ("run_phase", FIFO_item_rsp.convert2string_stimulus(),UVM_MEDIUM);
end
endtask

endclass //FIFO_monitor extends uvm_monitor

endpackage
```

## FIFO_sequencer :

```
package FIFO_sequencer_pkg ;
import uvm_pkg :: * ;
import FIFO_sequence_item_pkg ::*;
`include "uvm_macros.svh"

class FIFO_sequencer extends uvm_sequencer #(FIFO_sequence_item);
`uvm_component_utils(FIFO_sequencer);

function new(string name = "FIFO_sequencer " , uvm_component parent = null);
super.new(name , parent);
endfunction //new()
endclass //FIFO_sequencer extends uvm_sequencer #(FIFO_sequence_item)

endpackage
```

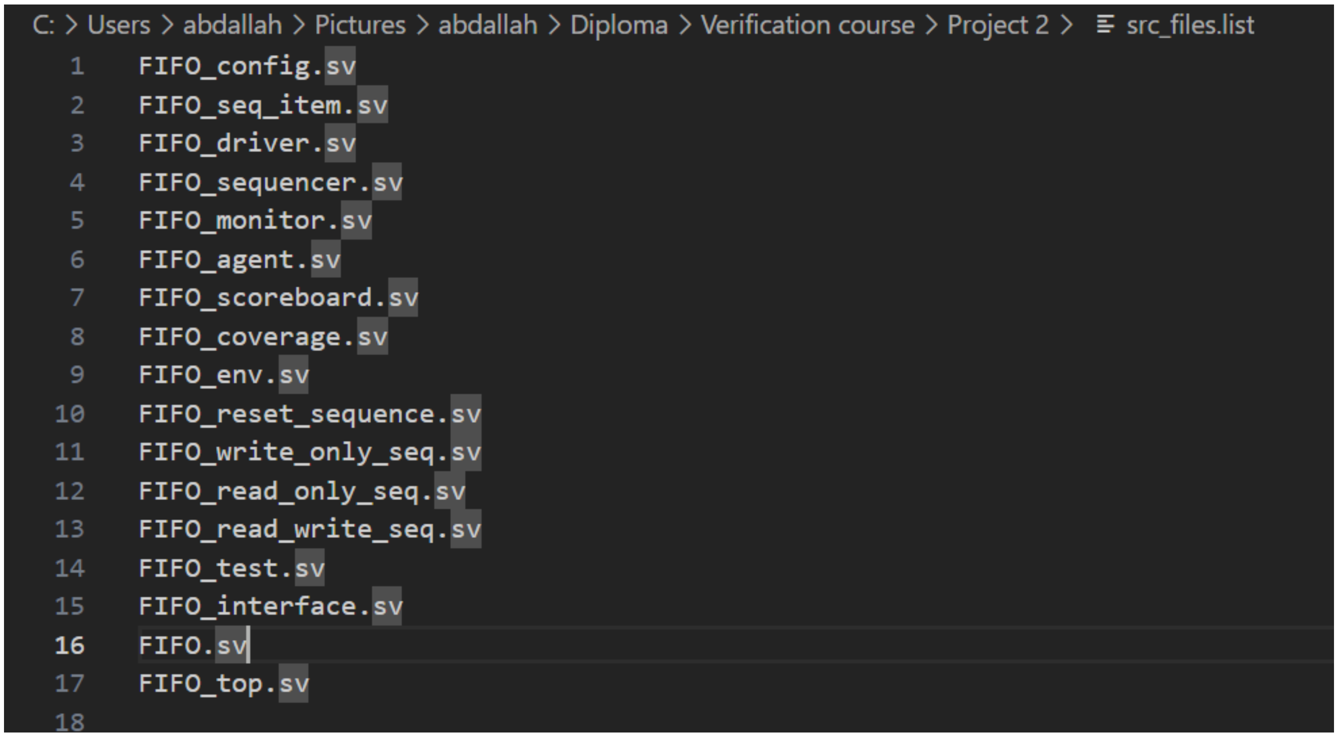
## FIFO_config_obj :

```
package FIFO_config_pkg;
import uvm_pkg :: * ;
`include "uvm_macros.svh"

class FIFO_config_obj extends uvm_object;
`uvm_object_utils(FIFO_config_obj)
virtual FIFO_interface FIFO_if;

function new(string name = " FIFO_config_obj");
super.new(name);
endfunction //new()

endclass //FIFO_config extends uvm_object
endpackage
```

## FIFO_sequence_item :

```
package FIFO_sequence_item_pkg ;
import uvm_pkg :: * ;
`include "uvm_macros.svh"

class FIFO_sequence_item extends uvm_sequence_item;
`uvm_object_utils(FIFO_sequence_item)

parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;
localparam max_fifo_addr = $clog2(FIFO_DEPTH);//put the local param before the signals
bit clk;
randc logic [FIFO_WIDTH-1:0] data_in;
rand logic 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 = 30 , WR_EN_ON_DIST = 70 ;

function new(string name = "FIFO_sequence_item");
super.new(name);
endfunction //new()

function string convert2string_stimulus();
return $sformatf("rst_n = 0b%b , wr_en = 0b%b , rd_en = 0b%b , data_in = 0b
%b , data_out = 0b%b , wr_ack = 0b%b , overflow = 0b%b , full = 0b%b , empty = 0b%b ,
almostfull = 0b%b , almostempty
= 0b%b , underflow = 0b%b " , rst_n, wr_en , rd_en , data_in , data_out, wr_ack,
overflow ,
full, empty, almostfull, almostempty, underflow );
endfunction

//Constraint blocks
constraint reset_signal { rst_n dist { 0:= 5 , 1:= 95 } ; } //reset contstraint
constraint write_enable_signal { wr_en dist { 0:= 100-WR_EN_ON_DIST , 1:=
WR_EN_ON_DIST } ; } //reset contstraint
constraint read_enable_signal { rd_en dist { 0:= 100-RD_EN_ON_DIST , 1:=
RD_EN_ON_DIST } ; } //reset contstraint

endclass //FIFO_sequence_item extends uvm_sequence_item

endpackage
```

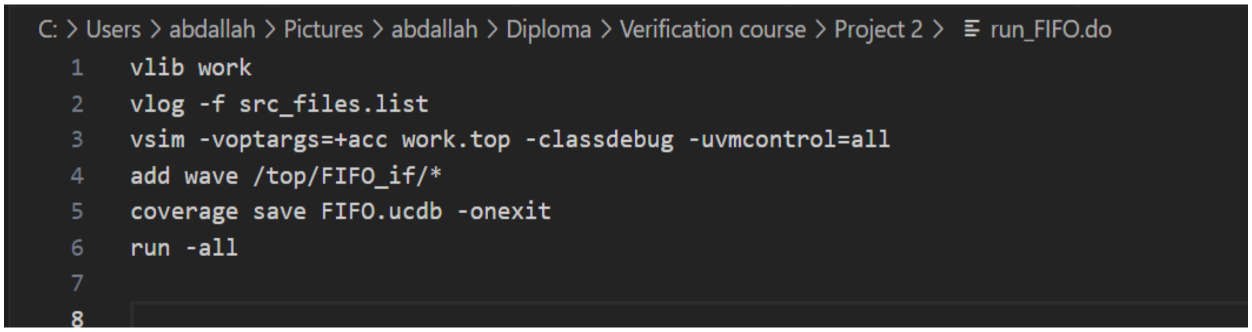
## src_files :



C: > Users > abdallah > Pictures > abdallah > Diploma > Verification course > Project 2 > src_files.list

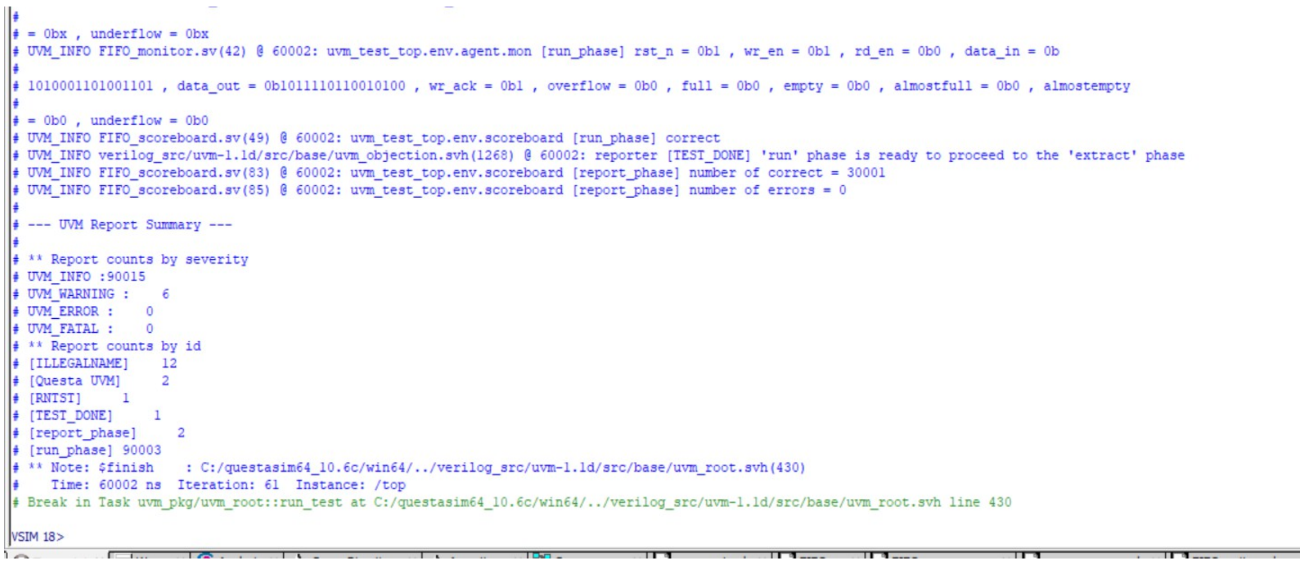
```
FIFO_config.sv
FIFO_seq_item.sv
FIFO_driver.sv
FIFO_sequencer.sv
FIFO_monitor.sv
FIFO_agent.sv
FIFO_scoreboard.sv
FIFO_coverage.sv
FIFO_env.sv
FIFO_reset_sequence.sv
FIFO_write_only_seq.sv
FIFO_read_only_seq.sv
FIFO_read_write_seq.sv
FIFO_test.sv
FIFO_interface.sv
FIFO.sv
FIFO_top.sv
```

## do file :


C: > Users > abdallah > Pictures > abdallah > Diploma > Verification course > Project 2 > run_FIFO.do

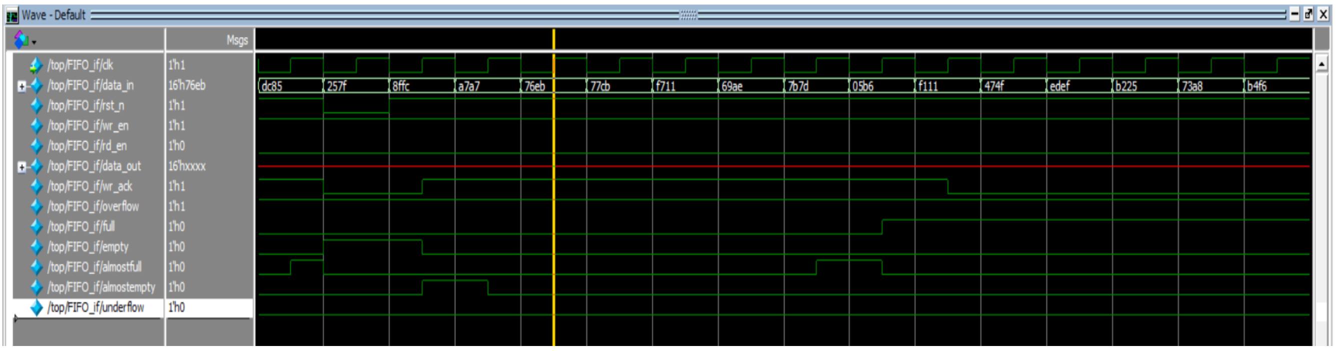
```
vlib work
vlog -f src_files.list
vsim -voptargs=+acc work.top -classdebug -uvmcontrol=all
add wave /top/FIFO_if/*
coverage save FIFO.ucdb -onexit
run -all
```

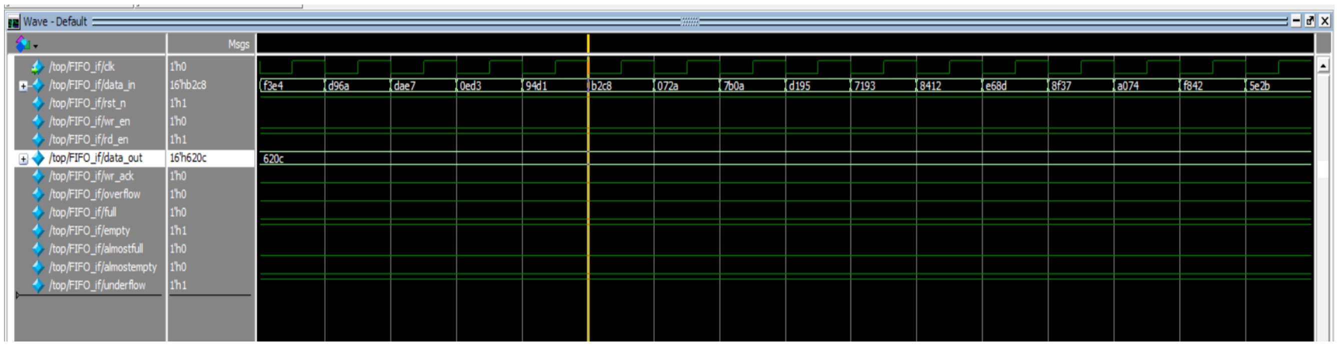
## trascript screenshot :



```
#
# = 0bx , underflow = 0bx
# UVM_INFO FIFO_monitor.sv(42) @ 60002: uvm_test_top.env.agent.mon [run_phase] rst_n = 0b1 , wr_en = 0b1 , rd_en = 0b0 , data_in = 0b
#
# 1010001101001101 , data_out = 0b1011110110010100 , wr_ack = 0b1 , overflow = 0b0 , full = 0b0 , empty = 0b0 , almostfull = 0b0 , almostempty
#
# = 0b0 , underflow = 0b0
# UVM_INFO FIFO_scoreboard.sv(49) @ 60002: uvm_test_top.env.scoreboard [run_phase] correct
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1268) @ 60002: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO FIFO_scoreboard.sv(83) @ 60002: uvm_test_top.env.scoreboard [report_phase] number of correct = 30001
# UVM_INFO FIFO_scoreboard.sv(85) @ 60002: uvm_test_top.env.scoreboard [report_phase] number of errors = 0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :90015
# UVM_WARNING :    6
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [ILLEGALNAME]    12
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# [report_phase]     2
# [run_phase] 90003
# ** Note: $finish    : C:/questasim64_10.6c/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 60002 ns  Iteration: 61  Instance: /top
# Break in Task uvm_pkg/uvm_root::run_test at C:/questasim64_10.6c/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh line 430

VSIM 18>
```

## simulations screenshot for the three sequences :

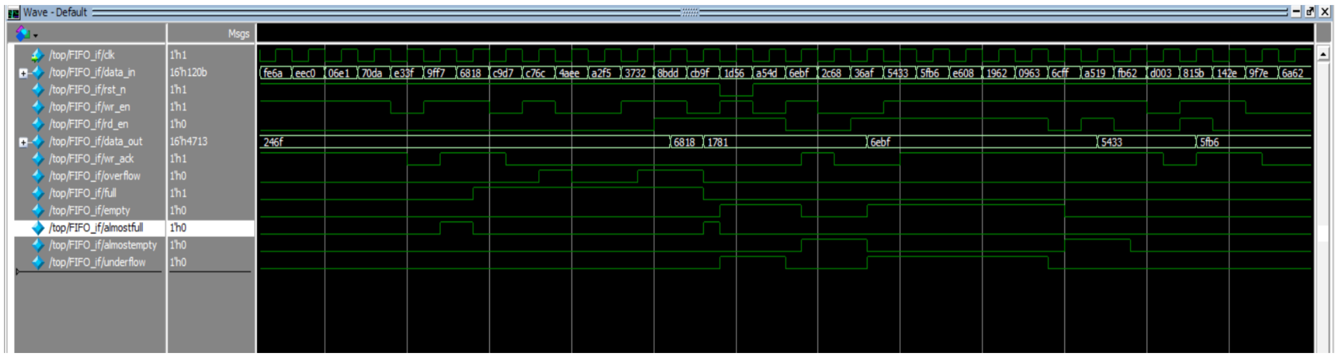




Wave - Default

```
                                 Msgs
/top/FIFO_if/clk                 1'h0
/top/FIFO_if/data_in             16'hb2c8
/top/FIFO_if/rst_n               1'h1
/top/FIFO_if/wr_en               1'h0
/top/FIFO_if/rd_en               1'h1
/top/FIFO_if/data_out            16'h620c
/top/FIFO_if/wr_ack              1'h0
/top/FIFO_if/overflow            1'h0
/top/FIFO_if/full                1'h0
/top/FIFO_if/empty               1'h1
/top/FIFO_if/almostfull          1'h0
/top/FIFO_if/almostempty         1'h0
/top/FIFO_if/underflow           1'h1
```

f3e4 d96a dae7 0ed3 94d1 b2c8 072a 7b0a d195 7193 8412 e68d 8f37 a074 f842 5e2b
620c

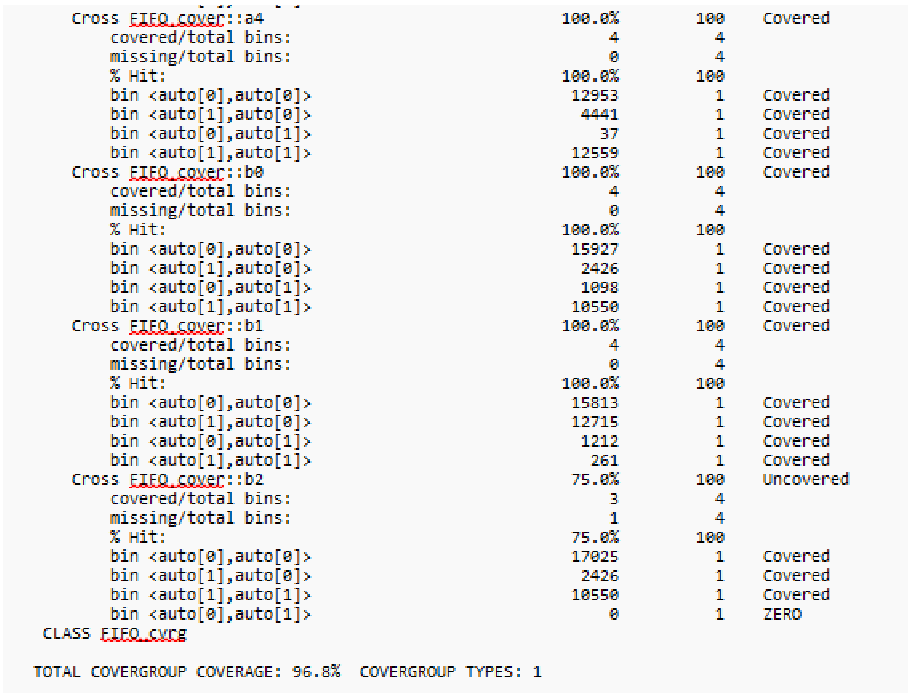


Wave - Default

```
                                 Msgs
/top/FIFO_if/clk                 1'h1
/top/FIFO_if/data_in             16'h120b
/top/FIFO_if/rst_n               1'h1
/top/FIFO_if/wr_en               1'h1
/top/FIFO_if/rd_en               1'h0
/top/FIFO_if/data_out            16'h4713
/top/FIFO_if/wr_ack              1'h1
/top/FIFO_if/overflow            1'h0
/top/FIFO_if/full                1'h1
/top/FIFO_if/empty               1'h0
/top/FIFO_if/almostfull          1'h0
/top/FIFO_if/almostempty         1'h0
/top/FIFO_if/underflow           1'h0
```

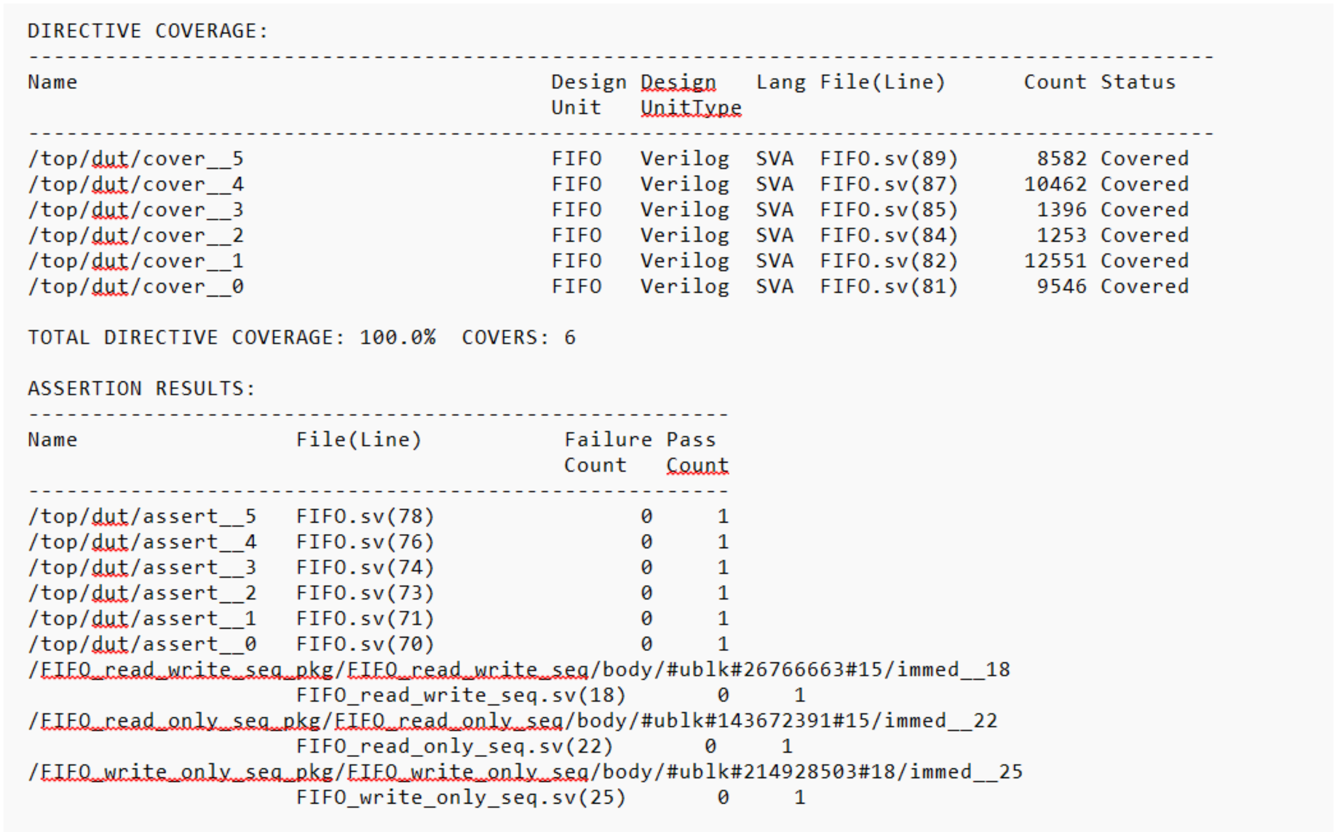
fe6a eec0 06e1 70da e33f 9ff7 6818 c9d7 c76c 4aee a2f5 3732 8bdd cb9f 1d56 a54d 6ebf 2c68 36af 5433 5fb6 e608 1962 0963 6cff a519 fb62 d003 815b 142e 9f7e 6a62
246f 6818 1781 6ebf 5433 5fb6

## functional coverage report :



```
    Cross FIFO_cover::a4                          100.0%        100    Covered
        covered/total bins:                            4          4
        missing/total bins:                            0          4
        % Hit:                                    100.0%        100
        bin <auto[0],auto[0]>                      12953          1    Covered
        bin <auto[1],auto[0]>                       4441          1    Covered
        bin <auto[0],auto[1]>                         37          1    Covered
        bin <auto[1],auto[1]>                      12559          1    Covered
    Cross FIFO_cover::b0                          100.0%        100    Covered
        covered/total bins:                            4          4
        missing/total bins:                            0          4
        % Hit:                                    100.0%        100
        bin <auto[0],auto[0]>                      15927          1    Covered
        bin <auto[1],auto[0]>                       2426          1    Covered
        bin <auto[0],auto[1]>                       1098          1    Covered
        bin <auto[1],auto[1]>                      10550          1    Covered
    Cross FIFO_cover::b1                          100.0%        100    Covered
        covered/total bins:                            4          4
        missing/total bins:                            0          4
        % Hit:                                    100.0%        100
        bin <auto[0],auto[0]>                      15813          1    Covered
        bin <auto[1],auto[0]>                      12715          1    Covered
        bin <auto[0],auto[1]>                       1212          1    Covered
        bin <auto[1],auto[1]>                        261          1    Covered
    Cross FIFO_cover::b2                           75.0%        100    Uncovered
        covered/total bins:                            3          4
        missing/total bins:                            1          4
        % Hit:                                     75.0%        100
        bin <auto[0],auto[0]>                      17025          1    Covered
        bin <auto[1],auto[0]>                       2426          1    Covered
        bin <auto[1],auto[1]>                      10550          1    Covered
        bin <auto[0],auto[1]>                          0          1    ZERO
 CLASS FIFO_cvrg

TOTAL COVERGROUP COVERAGE: 96.8%  COVERGROUP TYPES: 1
```

## assertion coverage :



```
DIRECTIVE COVERAGE:
--------------------------------------------------------------------------------------------
Name                                        Design Design   Lang File(Line)      Count Status
                                            Unit   UnitType
--------------------------------------------------------------------------------------------
/top/dut/cover__5                           FIFO   Verilog  SVA  FIFO.sv(89)      8582 Covered
/top/dut/cover__4                           FIFO   Verilog  SVA  FIFO.sv(87)     10462 Covered
/top/dut/cover__3                           FIFO   Verilog  SVA  FIFO.sv(85)      1396 Covered
/top/dut/cover__2                           FIFO   Verilog  SVA  FIFO.sv(84)      1253 Covered
/top/dut/cover__1                           FIFO   Verilog  SVA  FIFO.sv(82)     12551 Covered
/top/dut/cover__0                           FIFO   Verilog  SVA  FIFO.sv(81)      9546 Covered

TOTAL DIRECTIVE COVERAGE: 100.0%  COVERS: 6

ASSERTION RESULTS:
-------------------------------------------------------
Name                   File(Line)            Failure Pass
                                             Count   Count
-------------------------------------------------------
/top/dut/assert__5     FIFO.sv(78)                 0     1
/top/dut/assert__4     FIFO.sv(76)                 0     1
/top/dut/assert__3     FIFO.sv(74)                 0     1
/top/dut/assert__2     FIFO.sv(73)                 0     1
/top/dut/assert__1     FIFO.sv(71)                 0     1
/top/dut/assert__0     FIFO.sv(70)                 0     1
/FIFO_read_write_seq_pkg/FIFO_read_write_seq/body/#ublk#26766663#15/immed__18
                       FIFO_read_write_seq.sv(18)       0     1
/FIFO_read_only_seq_pkg/FIFO_read_only_seq/body/#ublk#143672391#15/immed__22
                       FIFO_read_only_seq.sv(22)       0     1
/FIFO_write_only_seq_pkg/FIFO_write_only_seq/body/#ublk#214928503#18/immed__25
                       FIFO_write_only_seq.sv(25)       0     1
```