FIFO PROJECT

FIFO_UVM Environment

NAME:SARA MAHMOUD ABDELWAHAB

Table of Contents

FIFO_UVM PROJECT

CODE

FIFO TOP

```
module FIFO_top ();
    import uvm_pkg::*;
`include "uvm_macros.svh"
import FIFO_TEST_pkg::*;

bit clk;
    initial
begin
    clk=0;
    forever
    begin
        #1 clk=~clk;
     end
end
FIFO_interface FIFO_if(clk);
FIFO dut(FIFO_if);
bind FIFO FIFO_SVA FIFO_SVA_IF(FIFO_if);
initial
begin
    uvm_config_db#(virtual FIFO_interface)::set(null,"uvm_test_top","FIFO_interface",FIFO_if);
    run_test("FIFO_test");
end


endmodule : FIFO_top
```

FIFO TEST

```
package FIFO_TEST_pkg;
import uvm_pkg::*;
import FIFO_ENV_pkg::*;
import FIFO_config_pkg::*;
import FIFO_sequences::*;
`include "uvm_macros.svh"
class FIFO_test extends uvm_test;
`uvm_component_utils(FIFO_test)
function  new(string name="FIFO_test", uvm_component parent =null);
    super.new(name,parent);
endfunction : new
FIFO_Env env;
FIFO_config_obj FIFO_config_obj_test;
FIFO_reset reset_seq;
FIFO_main_write  write_seq;
FIFO_main_read   read_seq;
FIFO_main_wr     wr_seq;

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_obj::type_id::create("FIFO_config_obj_test",this);
reset_seq=FIFO_reset::type_id::create("reset_seq",this);
write_seq=FIFO_main_write::type_id::create("write_seq",this);
read_seq=FIFO_main_read::type_id::create("read_seq",this);
wr_seq=FIFO_main_wr::type_id::create("wr_seq",this);
if(!uvm_config_db#(virtual FIFO_interface)::get(this,"","FIFO_interface",FIFO_config_obj_test.FIFO_config_vif))
    `uvm_error("build_phase","Test - Unable to get the virtual interface of the shifth reg from the uvm_config_db");

uvm_config_db #(FIFO_config_obj)::set(this,"*","FIFO_test_vif",FIFO_config_obj_test);
endfunction
task run_phase(uvm_phase phase);
   super.run_phase(phase);
    phase.raise_objection(this);
    `uvm_info("run_phase","Resset asserted.",UVM_LOW);
    reset_seq.start(env.agt.sqr);
    `uvm_info("run_phase","Resset deasserted.",UVM_LOW);

    `uvm_info("run_phase","stimulus generation started.",UVM_LOW);
    write_seq.start(env.agt.sqr);
    `uvm_info("run_phase","Resset stimulus generation Ended.",UVM_LOW);
    // phase.drop_objection(this);

    `uvm_info("run_phase","stimulus generation started.",UVM_LOW);
    read_seq.start(env.agt.sqr);
    `uvm_info("run_phase","Resset stimulus generation Ended.",UVM_LOW);
    // phase.drop_objection(this);

    `uvm_info("run_phase","stimulus generation started.",UVM_LOW);
    wr_seq.start(env.agt.sqr);
    `uvm_info("run_phase","Resset stimulus generation Ended.",UVM_LOW);
    phase.drop_objection(this);
endtask : run_phase
    endclass : FIFO_test
endpackage : FIFO_TEST_pkg
```

FIFO SEQUENCES

```
package FIFO_sequences;
import uvm_pkg::*;
import FIFO_seq_item_pkg::*;
import FIFO_sequencer_pkg::*;
import shared_pkg::*;
`include "uvm_macros.svh"
class FIFO_reset extends  uvm_sequence #(FIFO_seq_item);
  `uvm_object_utils(FIFO_reset);
  FIFO_seq_item seq_item_reset;
function  new(string name ="FIFO_reset");
  super.new(name);
  endfunction : new
task body();
seq_item_reset =FIFO_seq_item::type_id::create("seq_item_reset");
start_item(seq_item_reset);
seq_item_reset.data_in=0;
seq_item_reset.rst_n=0;
seq_item_reset.rd_en=0;
seq_item_reset.wr_en=0;
finish_item(seq_item_reset);
endtask : body
endclass : FIFO_reset


class FIFO_main_write extends  uvm_sequence #(FIFO_seq_item);
  `uvm_object_utils(FIFO_main_write);
  FIFO_seq_item write_seq;

function  new(string name ="FIFO_main_write");
  super.new(name);
endfunction : new

task body();
     repeat(1000) begin
     write_seq = FIFO_seq_item::type_id::create("write_seq");
     start_item(write_seq);
    // Randomize with rd_en fixed to 0
            write_seq.randomize() with { write_seq.rd_en == 0; write_seq.wr_en==1;};
     finish_item(write_seq);
     end

endtask : body

endclass : FIFO_main_write

```

```
class FIFO_main_write extends  uvm_sequence #(FIFO_seq_item);
  `uvm_object_utils(FIFO_main_write);
  FIFO_seq_item write_seq;

function  new(string name ="FIFO_main_write");
  super.new(name);
endfunction : new

task body();
     repeat(1000) begin
     write_seq = FIFO_seq_item::type_id::create("write_seq");
     start_item(write_seq);
            write_seq.randomize() with { write_seq.rd_en == 0; write_seq.wr_en==1;};
     finish_item(write_seq);
     end
endtask : body
endclass : FIFO_main_write
class FIFO_main_read extends  uvm_sequence #(FIFO_seq_item);
  `uvm_object_utils(FIFO_main_read);
  FIFO_seq_item seq_item_main;
    FIFO_seq_item read_seq;
function  new(string name ="FIFO_main_read");
  super.new(name);
endfunction : new
task body();
        repeat(1000) begin
        read_seq = FIFO_seq_item::type_id::create("read_seq");
        start_item(read_seq);
           read_seq.randomize() with { read_seq.rd_en == 1; read_seq.wr_en==0;};
       finish_item(read_seq);
      end
endtask : body
endclass : FIFO_main_read
class FIFO_main_wr extends  uvm_sequence #(FIFO_seq_item);
  `uvm_object_utils(FIFO_main_wr);
  FIFO_seq_item seq_item_main;
  FIFO_seq_item write_seq;
    FIFO_seq_item read_seq;
    FIFO_seq_item wr_seq;
function  new(string name ="FIFO_main_wr");
  super.new(name);
endfunction : new
task body();
     repeat(1000) begin
        wr_seq = FIFO_seq_item::type_id::create("wr_seq");
        start_item(wr_seq);
        assert(wr_seq.randomize());
       finish_item(wr_seq);
     end
endtask : body
endclass : FIFO_main_wr
endpackage : FIFO_sequences
```

FIFO Environment

```
package FIFO_ENV_pkg;
import uvm_pkg::*;
import FIFO_driver_pkg::*;
import FIFO_sequencer_pkg::*;
import FIFO_agent_pkg::*;
import FIFO_scoreboard_pkg::*;
import FIFO_coverage_pkg::*;


`include "uvm_macros.svh"

class FIFO_Env extends  uvm_env;
    `uvm_component_utils(FIFO_Env);

FIFO_driver driver;
FIFO_sequencer FIFO_sqr;
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 : build_phase

function void connect_phase(uvm_phase phase) ;
    super.connect_phase(phase);
    //driver.seq_item_port.connect(FIFO_sqr.seq_item_export);
    agt.agt_ap.connect(sb.sb_export);
    agt.agt_ap.connect(cov.cov_export);

endfunction


endclass : FIFO_Env


endpackage : FIFO_ENV_pkg

```

FIFO COVERAGE COLLECTOR

```
package FIFO_coverage_pkg;
import uvm_pkg::*;
import FIFO_driver_pkg::*;
import FIFO_seq_item_pkg::*;
import FIFO_sequencer_pkg::*;
import FIFO_config_pkg::*;
import FIFO_monitor_pkg::*;
import shared_pkg::*;

`include "uvm_macros.svh"
 class FIFO_coverage extends  uvm_component;
   `uvm_component_utils(FIFO_coverage);
   uvm_analysis_export #(FIFO_seq_item) cov_export;
    uvm_tlm_analysis_fifo #(FIFO_seq_item) cov_fifo;
     FIFO_seq_item seq_item_cov;
 covergroup FIFO_coverage_gp ;
   wr_ack:coverpoint seq_item_cov.wr_ack{
   bins wr_ack_1={1};
   bins wr_ack_0={0};
   }
   wr_en:coverpoint seq_item_cov.wr_en{
   bins wr_en_1={1};
   bins wr_en_0={0};
   }
   rd_en:coverpoint seq_item_cov.rd_en{
   bins rd_en_1={1};
   bins rd_en_0={0};
   }
   full:coverpoint seq_item_cov.full{
   bins full_1={1};
   bins full_0={0};
   }
   underflow:coverpoint seq_item_cov.underflow{
   bins underflow_1={1};
   bins underflow_0={0};
   }
   overflow:coverpoint seq_item_cov.overflow{
   bins overflow_1={1};
   bins overflow_0={0};
   }
    // Cross coverage of wr_en, rd_en, and all control signals
   empty_cv:cross seq_item_cov.wr_en, seq_item_cov.rd_en, seq_item_cov.empty;
   almostfull_cv:cross seq_item_cov.wr_en, seq_item_cov.rd_en, seq_item_cov.almostfull;
   almostempty_cv:cross seq_item_cov.wr_en, seq_item_cov.rd_en, seq_item_cov.almostempty ;
    wr_ack_cv: cross wr_en, rd_en, wr_ack {
          ignore_bins wr_ack_cv_excl_0_1_1 = binsof(wr_en.wr_en_0)&&binsof(rd_en.rd_en_1)&&binsof(wr_ack.wr_ack_1);
          ignore_bins wr_ack_cv_excl_0_0_1 = binsof(wr_en.wr_en_0)&&binsof(rd_en.rd_en_0)&&binsof(wr_ack.wr_ack_1);
      }

     overflow_cv: cross wr_en, rd_en, overflow {
          ignore_bins overflow_cv_excl_0_1_1 = binsof(wr_en.wr_en_0)&&binsof(rd_en.rd_en_1)&&binsof(overflow.overflow_1);
          ignore_bins overflow_cv_excl_0_0_1 = binsof(wr_en.wr_en_0)&&binsof(rd_en.rd_en_0)&&binsof(overflow.overflow_1);
      }

```

```
54
55          full_cv: cross wr_en, rd_en, full {
56              ignore_bins full_cv_excl_1_1_1 = binsof(wr_en.wr_en_1)&&binsof(rd_en.rd_en_1)&&binsof(full.full_1);
57              ignore_bins full_cv_excl_0_1_1 = binsof(wr_en.wr_en_0)&&binsof(rd_en.rd_en_1)&&binsof(full.full_1);
58          }
59
60          underflow_cv: cross wr_en, rd_en, underflow {
61              ignore_bins underflow_cv_excl_1_0_1 = binsof(wr_en.wr_en_1)&&binsof(rd_en.rd_en_0)&&binsof(underflow.underflow_1);
62              ignore_bins underflow_cv_excl_0_0_1 = binsof(wr_en.wr_en_0)&&binsof(rd_en.rd_en_0)&&binsof(underflow.underflow_1);
63          }
64
65      endgroup : FIFO_coverage_gp
66              function  new(string name="FIFO_coverage_gp", uvm_component parent =null);
67        super.new(name,parent);
68        FIFO_coverage_gp=new();
69      endfunction : new
70
71        function void build_phase(uvm_phase phase);
72              super.build_phase(phase);
73               cov_export=new("cov_export",this);
74               cov_fifo=new("cov_fifo",this);
75               endfunction
76
77           function void connect_phase(uvm_phase phase) ;
78            super.connect_phase(phase);
79            cov_export.connect(cov_fifo.analysis_export);
80             endfunction
81
82            task run_phase(uvm_phase phase);
83             super.run_phase(phase);
84             forever
85             begin
86               cov_fifo.get(seq_item_cov);
87               FIFO_coverage_gp.sample();
88             end
89
90            endtask : run_phase
91
92    endclass : FIFO_coverage
93
94    endpackage : FIFO_coverage_pkg
```

FIFO SCOREBOARD

```
package FIFO_scoreboard_pkg;
import uvm_pkg::*;
import FIFO_driver_pkg::*;
import FIFO_seq_item_pkg::*;
import FIFO_sequencer_pkg::*;
import FIFO_config_pkg::*;
import FIFO_monitor_pkg::*;
import shared_pkg::*;
`include "uvm_macros.svh"
  class FIFO_scoreboard extends uvm_scoreboard;
   `uvm_component_utils(FIFO_scoreboard)
   uvm_analysis_export #(FIFO_seq_item) sb_export;
   uvm_tlm_analysis_fifo #(FIFO_seq_item) sb_fifo;
   FIFO_seq_item seq_item_sb;

    logic [FIFO_WIDTH-1:0] data_in_ref;
    logic rst_n_ref, wr_en_ref, rd_en_ref;
    logic [FIFO_WIDTH-1:0] data_out_ref;
    logic wr_ack_ref, overflow_ref;
    logic full_ref, empty_ref, almostfull_ref, almostempty_ref, underflow_ref,write_read;
    reg [2:0] write_pointer=0;
    reg [2:0] read_pointer=0;
    integer  fifo_count = 0;
    reg [FIFO_WIDTH-1:0] mem_ref [FIFO_DEPTH-1:0];
function  new(string name="FIFO_scoreboard", uvm_component parent =null);
    super.new(name,parent);
    this.write_pointer=0;
      this.read_pointer=0;
      this.fifo_count=0;
  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(data_out_ref ===seq_item_sb.data_out)
    begin
      `uvm_info("run_phase",$sformatf("correct DATA OUT:%S",seq_item_sb.convert2string()),UVM_HIGH);
      correct_count++;
  end
else
  begin
      `uvm_error("run_phase",$sformatf("comparison failed,transaction received by the DUT:%s while the reference data_out:0h%0h",seq_item_sb.convert2string(),data_out_ref));
      error_count++;
    end
  end
```



```
  task ref_model(FIFO_seq_item seq_item_chk);
        data_in_ref=seq_item_chk.data_in;
        rst_n_ref=seq_item_chk.rst_n;
        wr_en_ref=seq_item_chk.wr_en;
          rd_en_ref=seq_item_chk.rd_en;
         write_read=wr_en_ref && rd_en_ref &&write_pointer==read_pointer;
         // Implement the golden model logic based on FIFO behavior
              // Calculate full_ref, empty_ref, almostfull_ref, almostempty_ref based on fifo_count
        full_ref       = (fifo_count == 8);    // Full when count reaches 8
        empty_ref      = (fifo_count == 0);    // Empty when count is 0
        almostfull_ref = (fifo_count == 6);    // Almost full at 7
        almostempty_ref = (fifo_count == 1);   // Almost empty at 1
        underflow_ref  = (rd_en_ref && empty_ref);  // Underflow condition
  // Update fifo_count based on wr_en and rd_en conditions
        if(!rst_n_ref)
          fifo_count=0;
        else if (wr_en_ref && !rd_en_ref && !full_ref) begin
          fifo_count++;
        end
        else if (rd_en_ref && !wr_en_ref && !empty_ref) begin
          fifo_count--;
        end
      else if ( ({wr_en_ref, rd_en_ref} == 2'b11) && empty_ref)
         fifo_count++;
      else if ( ({wr_en_ref, rd_en_ref} == 2'b11) && full_ref)
        fifo_count--;
        // Write operation: Check if write enable is high and FIFO is not full
       if (!rst_n_ref) begin
    write_pointer = 0;
    read_pointer = 0;
  end else begin
    // Handle simultaneous write and read when write_pointer == read_pointer
    if (wr_en_ref && rd_en_ref && (write_pointer == read_pointer) && !full_ref && !empty_ref) begin
          // Read the old data before it gets overwritten
      data_out_ref = mem_ref[read_pointer];
      // Write the new data
      mem_ref[write_pointer] = data_in_ref;

      // Increment both pointers (wrap them around using modulo logic)
      write_pointer = (write_pointer + 1) ;
      read_pointer = (read_pointer + 1) ;
    end else begin
      // Write operation: Check if write enable is high and FIFO is not full
      if (wr_en_ref && !full_ref) begin
        mem_ref[write_pointer] = data_in_ref;
        write_pointer = (write_pointer + 1) ;  // Increment write pointer with wrap around
      end
      // Read operation: Check if read enable is high and FIFO is not empty
      if (rd_en_ref && !empty_ref) begin
        data_out_ref = mem_ref[read_pointer];  // Provide the data for read
        read_pointer = (read_pointer + 1) ;  // Increment read pointer with wrap around
      end
    end
  end
    endtask: ref_model
```

```
    function void report_phase(uvm_phase phase);
        super.report_phase(phase);
        `uvm_info("report_phase",$sformatf("total successful transaction:%0d",correct_count),UVM_MEDIUM);
          `uvm_info("reprt_phase",$sformatf("total failed transaction:%0d",error_count),UVM_MEDIUM);
      endfunction : report_phase
    endclass : FIFO_scoreboard

  endpackage : FIFO_scoreboard_pkg
```

FIFO AGENT

```
package FIFO_agent_pkg;
import uvm_pkg::*;
import FIFO_driver_pkg::*;
import FIFO_coverage_pkg::*;
import FIFO_seq_item_pkg::*;
import FIFO_sequencer_pkg::*;
import FIFO_config_pkg::*;
import FIFO_monitor_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 FIFO_cfg;
        uvm_analysis_port #(FIFO_seq_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,"*","FIFO_test_vif",FIFO_cfg))
begin
    `uvm_fatal("build_phase"," Unable to get the configuration object");
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) ;
    super.connect_phase(phase);
    drv.FIFO_driver_vif=FIFO_cfg.FIFO_config_vif;
    mon.FIFO_vif=FIFO_cfg.FIFO_config_vif;
    drv.seq_item_port.connect(sqr.seq_item_export);
    mon.mon_ap.connect(agt_ap);
endfunction


    endclass : FIFO_agent


endpackage : FIFO_agent_pkg
```

FIFO SEQUENCER

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

class FIFO_sequencer extends  uvm_sequencer #(FIFO_seq_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

endpackage : FIFO_sequencer_pkg
```

FIFO SEQUENCE ITEM

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

class FIFO_seq_item extends  uvm_sequence_item;
`uvm_object_utils(FIFO_seq_item)
parameter FIFO_WIDTH = 16;
Logic clk;
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;

 // Integers for distribution percentages
 integer RD_EN_ON_DIST;
 integer WR_EN_ON_DIST;

 function  new(string name ="FIFO_seq_item",int rd_dist = 30, int wr_dist = 70);
    super.new(name);
    this.RD_EN_ON_DIST = rd_dist;
    this.WR_EN_ON_DIST = wr_dist;
endfunction : new

function string convert2string();
    return $sformatf("%s rst =0b%0b, data_in=0h%0h , wr_en=0b%0b , rd_en=0b%0b , data_out=0h%0h ,wr_ack=0b%0b , overflow=0b%0b , full=0b%0b , empty=0b%0b ,almostfull=0b%0b , almostem
        rst_n,data_in, wr_en, rd_en, data_out, wr_ack,overflow,full, empty, almostfull,almostempty,underflow);

endfunction : convert2string

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

endfunction : convert2string_stimulus

    constraint rst_transaction_const {
      rst_n dist {1'b1 :=98, 1'b0 := 2};  // 90% chance of being deactivated (active low reset)
    }

     constraint wr_transaction_const {
      wr_en dist {1 := WR_EN_ON_DIST, 0 := (100 - WR_EN_ON_DIST)};
    }

     constraint rd_transaction_const {
      rd_en dist {1 := RD_EN_ON_DIST, 0 := (100 - RD_EN_ON_DIST)};
    }

endclass : FIFO_seq_item
endpackage : FIFO_seq_item_pkg
```


FIFO MONITOR

```
package FIFO_monitor_pkg;
import uvm_pkg::*;
import FIFO_seq_item_pkg::*;
import shared_pkg::*;

`include "uvm_macros.svh"

    class FIFO_monitor extends  uvm_monitor;
        `uvm_component_utils(FIFO_monitor)
        virtual FIFO_interface FIFO_vif;
        FIFO_seq_item rsp_seq_item;
        uvm_analysis_port #(FIFO_seq_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 : build_phase

task run_phase(uvm_phase phase);
super.run_phase(phase);
forever
begin
rsp_seq_item=FIFO_seq_item::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.data_out=FIFO_vif.data_out;
rsp_seq_item.wr_ack=FIFO_vif.wr_ack;
rsp_seq_item.overflow=FIFO_vif.overflow;
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.underflow=FIFO_vif.underflow;
mon_ap.write(rsp_seq_item);
`uvm_info("run_phase",rsp_seq_item.convert2string_stimulus(),UVM_HIGH)
end

endtask : run_phase


    endclass : FIFO_monitor
endpackage : FIFO_monitor_pkg
```

FIFO DRIVER

```
package FIFO_driver_pkg;
import uvm_pkg::*;
import FIFO_config_pkg::*;
import FIFO_seq_item_pkg::*;



`include "uvm_macros.svh"
class FIFO_driver extends  uvm_driver #(FIFO_seq_item);
`uvm_component_utils(FIFO_driver)
virtual FIFO_interface FIFO_driver_vif;
FIFO_config_obj FIFO_config_obj_driver;
FIFO_seq_item stim_FIFO_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_FIFO_seq_item=FIFO_seq_item::type_id::create("stim_FIFO_seq_item");
seq_item_port.get_next_item(stim_FIFO_seq_item);
FIFO_driver_vif.data_in = stim_FIFO_seq_item.data_in; FIFO_driver_vif.rst_n = stim_FIFO_seq_item.rst_n; FIFO_driver_vif.wr_en = stim_FIFO_seq_item.wr_en;
FIFO_driver_vif.rd_en = stim_FIFO_seq_item.rd_en;
 @(negedge FIFO_driver_vif.clk);
seq_item_port.item_done();
`uvm_info("run_phase",stim_FIFO_seq_item.convert2string_stimulus(),UVM_HIGH)
end

endtask: run_phase

endclass : FIFO_driver

endpackage : FIFO_driver_pkg
```

FIFO CONFIGURATION

```
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_config_vif;

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

endpackage : FIFO_config_pkg
```

FIFO INTERFACE

```
interface FIFO_interface (
input bit 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;

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_interface
```

FIFO DESIGN

```
module FIFO(FIFO_interface.DUT FIFO_if);
parameter FIFO_WIDTH = 16;
parameter FIFO_DEPTH = 8;
localparam max_fifo_addr = $clog2(FIFO_DEPTH);

reg [FIFO_WIDTH-1:0] mem [FIFO_DEPTH-1:0];

reg [max_fifo_addr-1:0] wr_ptr, rd_ptr;
reg [max_fifo_addr:0] count;

always @(posedge FIFO_if.clk or negedge FIFO_if.rst_n) begin
  if (!FIFO_if.rst_n) begin
    wr_ptr <= 0;
    FIFO_if.wr_ack<=0;
    FIFO_if.overflow<=0;
  end
  else 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

always @(posedge FIFO_if.clk or negedge FIFO_if.rst_n) begin
  if (!FIFO_if.rst_n) begin
    rd_ptr <= 0;
  end
  else if (FIFO_if.rd_en && !FIFO_if.empty) begin
    FIFO_if.data_out <= mem[rd_ptr];
    rd_ptr <= rd_ptr + 1;
  end
end

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.rd_en} == 2'b10) && !FIFO_if.full)
      count <= count + 1;
    else if ( ({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b01) && !FIFO_if.empty)
      count <= count - 1;
    else if ( ({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b11) && FIFO_if.empty)
       count <= count + 1;
    else if ( ({FIFO_if.wr_en, FIFO_if.rd_en} == 2'b11) && FIFO_if.full)
      count <= count - 1;
  end
end
```

```
always@(posedge FIFO_if.clk or negedge FIFO_if.rst_n) begin
  if (!FIFO_if.rst_n)
    FIFO_if.underflow<=0;
    else if(FIFO_if.empty && FIFO_if.rd_en)
    FIFO_if.underflow<=1;
  else
    FIFO_if.underflow<=0;
  end

assign FIFO_if.full = (count == FIFO_DEPTH)? 1 : 0;
assign FIFO_if.empty = (count == 0)? 1 : 0;
//assign FIFO_if.underflow = (FIFO_if.empty && FIFO_if.rd_en)? 1 : 0;
assign FIFO_if.almostfull = (count == FIFO_DEPTH-2)? 1 : 0;
assign FIFO_if.almostempty = (count == 1)? 1 : 0;



endmodule
```

FIFO SVA

```
import shared_pkg::*;

module FIFO_SVA(FIFO_interface.DUT FIFO_if);
    // ASSERTIONS
  // Count Validity Property: Ensure that count stays within valid limits
  property count_valid;
    @(posedge FIFO_if.clk) disable iff(FIFO_if.rst_n==0) (FIFO_top.dut.count <= FIFO_DEPTH);
  endproperty

  assert_count_valid: assert property(count_valid) else $error("Error: FIFO count exceeds FIFO_DEPTH");
  cover_count_valid: cover property(count_valid);

  // Overflow Condition Property: Overflow only happens when FIFO_if.full and write is enabled
  property overflow_cond;
    @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.full && FIFO_if.wr_en)|=>FIFO_if.overflow==1;
  endproperty

  assert_overflow_cond: assert property(overflow_cond) else $error("Error: Overflow assertion failed");
  cover_overflow_cond: cover property(overflow_cond);

  // Underflow Condition Property: Underflow only happens when FIFO_if.empty and read is enabled
  property underflow_cond;
    @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.empty && FIFO_if.rd_en)|=>(FIFO_if.underflow ==1);
  endproperty

  assert_underflow_cond: assert property(underflow_cond) else $error("Error: Underflow assertion failed");
  cover_underflow_cond: cover property(underflow_cond);

  // Full Flag Property: Full flag is set when count equals FIFO_DEPTH
  property full_cond;
    @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.full == (FIFO_top.dut.count == FIFO_DEPTH));
  endproperty

  assert_full_cond: assert property(full_cond) else $error("Error: Full flag assertion failed");
  cover_full_cond: cover property(full_cond);

  // Empty Flag Property: Empty flag is set when count equals 0
  property empty_cond;
    @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.empty == (FIFO_top.dut.count == 0));
  endproperty

  assert_empty_cond: assert property(empty_cond) else $error("Error: Empty flag assertion failed");
  cover_empty_cond: cover property(empty_cond);

  // Almost Full Flag Property: Almost FIFO_if.full flag is set when count equals FIFO_DEPTH-2
  property almostfull_cond;
    @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.almostfull == (FIFO_top.dut.count == FIFO_DEPTH-2));
  endproperty

  assert_almostfull_cond: assert property(almostfull_cond) else $error("Error: Almost FIFO_if.full flag assertion failed");
  cover_almostfull_cond: cover property(almostfull_cond);

  // Almost Empty Flag Property: Almost FIFO_if.empty flag is set when count equals 1
  property almostempty_cond;
    @(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n) (FIFO_if.almostempty == (FIFO_top.dut.count == 1));
  endproperty

  assert_almostempty_cond: assert property(almostempty_cond) else $error("Error: Almost FIFO_if.empty flag assertion failed");
  cover_almostempty_cond: cover property(almostempty_cond);

  // Write Pointer Boundaries Property: Write pointer is always within bounds
  property wr_ptr_valid;
    @(posedge FIFO_if.clk) (FIFO_top.dut.wr_ptr < FIFO_DEPTH);
  endproperty

  assert_wr_ptr_valid: assert property(wr_ptr_valid) else $error("Error: Write pointer out of bounds");
  cover_wr_ptr_valid: cover property(wr_ptr_valid);

  // Read Pointer Boundaries Property: Read pointer is always within bounds
  property rd_ptr_valid;
    @(posedge FIFO_if.clk) (FIFO_top.dut.rd_ptr < FIFO_DEPTH);
  endproperty

  assert_rd_ptr_valid: assert property(rd_ptr_valid) else $error("Error: Read pointer out of bounds");
  cover_rd_ptr_valid: cover property(rd_ptr_valid);


endmodule : FIFO_SVA
```

FIFO VERIFICATION PLAN

Label	Design Requirement Description	Stimulus Generation	Functional Coverage	Functionality Check
FIFO_1	When the reset is asserted, all FIFO signals should be reset to their default values	Reset signal rst_n is randomized with a distribution favoring the deasserted state (98% low, 2% high)		Scoreboard checks to check for the reset functionality
FIFO_2	When wr_en is asserted, data should be written to the FIFO	Randomized wr_en signal driven by a 70% chance of being enabled, along with randomized data_in	Cross coverage of wr_en, rd_en, and wr_ack ensures all write operations	Scoreboard checks to verify correct data is written
FIFO_3	When rd_en is asserted, the FIFO should output the correct data	Randomized rd_en signal driven by a 30% chance of being enabled	Cross coverage of wr_en, rd_en, and empty ensures all read scenarios	Scoreboard checks to verify correct data is read(data_out)
FIFO_4	The overflow signal should be asserted when writing to a full FIFO	Randomized write operations until the FIFO becomes full, using wr_en distribution	Cross coverage of wr_en, rd_en, and overflow signals	Concurrent assertion to check for correct overflow signal
FIFO_5	The underflow signal should be asserted when reading from an empty FIFO	Randomized read operations until the FIFO becomes empty, using rd_en distribution	Cross coverage of wr_en, rd_en, and underflow	Concurrent assertion to check for correct underflow signal
FIFO_6	The full signal should be asserted when the FIFO is full		Cross coverage of wr_en, rd_en, and full	Concurrent assertion to check for correct full signal assertion
FIFO_7	The empty signal should be asserted when the FIFO is empty		Cross coverage of wr_en, rd_en, and empty	Concurrent assertion to check for correct empty signal assertion
FIFO_8	The almostfull signal should be asserted when the FIFO is nearly full		Cross coverage of wr_en, rd_en, and almostfull	Concurrent assertion to check almostfull behavior
FIFO_9	The almostempty signal should be asserted when the FIFO is nearly empty		Cross coverage of wr_en, rd_en, and almostemoty	Concurrent assertion to check almostempty behavior
FIFO_10	Ensure data integrity between data_in and data_out through correct FIFO operation			Scoreboard checks for data integrity across all transactions
FIFO_11	Ensure that wr_ack signal is correctly asserted after a valid write operation		Cross coverage of wr_en, rd_en, and wr_ack ensures all write operations	Concurrent assertion to verify correct wr_ack signal

DO & SOURCE FILES

```
vlib work
vlog -f src_files.list +cover -covercells
vsim -voptargs=+acc work.FIFO_top -classdebug -uvmcontrol=all  -cover
add wave /FIFO_top/FIFO_if/*
coverage save FIFO_top.ucdb -onexit
run -all
coverage exclude -cvgpath {/FIFO_coverage_pkg/FIFO_coverage/FIFO_coverage_gp/wr_ack_cv/<auto[0],auto[1],auto[1]>} {/FIFO_coverage_pkg/FIFO_coverage/FIFO_coverage_gp/wr_ack_cv/<auto[0],auto[0],auto[1]>}
coverage exclude -cvgpath {/FIFO_coverage_pkg/FIFO_coverage/FIFO_coverage_gp/overflow_cv/<auto[0],auto[1],auto[1]>} {/FIFO_coverage_pkg/FIFO_coverage/FIFO_coverage_gp/overflow_cv/<auto[0],auto[0],auto[1]>}
coverage exclude -cvgpath {/FIFO_coverage_pkg/FIFO_coverage/FIFO_coverage_gp/full_cv/<auto[1],auto[1],auto[1]>} {/FIFO_coverage_pkg/FIFO_coverage/FIFO_coverage_gp/full_cv/<auto[0],auto[1],auto[1]>}
coverage exclude -cvgpath {/FIFO_coverage_pkg/FIFO_coverage/FIFO_coverage_gp/underflow_cv/<auto[1],auto[0],auto[1]>} {/FIFO_coverage_pkg/FIFO_coverage/FIFO_coverage_gp/underflow_cv/<auto[0],auto[0],auto[1]>}

```

```
FIFO.sv
FIFO_agent_pkg.sv
FIFO_config_pkg.sv
FIFO_coverage_pkg.sv
FIFO_driver_pkg.sv
FIFO_ENV_pkg.sv
FIFO_monitor_pkg.sv
FIFO_scoreboard_pkg.sv
FIFO_sequencer_pkg.sv
FIFO_sequences.sv
FIFO_seq_item_pkg.sv
FIFO_SVA.sv
FIFO_TEST_pkg.sv
FIFO_top.sv
interface.sv
shared_pkg.sv

```

BUG REPORT

In the FIFO design, I addressed several critical bugs and improved the functionality. First, I changed the underflow signal from combinational to sequential, ensuring that it is updated based on the state of the FIFO during the clock edge.Additionally, I implemented a priority system where, if both write enable (wr_en) and read enable (rd_en) are high, the FIFO prioritizes writing when it is empty and reading when it is full. To manage the count effectively, I utilized the full and empty flags instead of directly comparing count, ensuring accurate updates to the count based on the current operations. These changes significantly enhance the robustness and functionality of the FIFO design. Furthermore, I reset the wr_ack and overflow flags.

COVERAGE REPORT

CODE COVERAGE

```
=== Instance: /FIFO_top/dut
=== Design Unit: work.FIFO
================================================================================
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        26        26         0   100.00%

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

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

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

Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                      28        28         0   100.00%

==============================Statement Details==============================
```

```
================================Toggle Details================================

Toggle Coverage for instance /FIFO_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)


=============================================================================
=== Instance: /FIFO_top/FIFO_if
=== Design Unit: work.FIFO_interface
=============================================================================
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         86        86         0   100.00%

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

Toggle Coverage for instance /FIFO_top/FIFO_if --

                                            Node      1H->0L      0L->1H  "Coverage"
                                            ----------------------------------------
                                      almostempty           1           1      100.00
                                       almostfull           1           1      100.00
                                              clk           1           1      100.00
                                     data_in[15-0]          1           1      100.00
                                    data_out[15-0]          1           1      100.00
                                             empty          1           1      100.00
                                              full          1           1      100.00
                                          overflow          1           1      100.00
                                             rd_en          1           1      100.00
                                             rst_n          1           1      100.00
                                         underflow          1           1      100.00
                                            wr_ack          1           1      100.00
                                             wr_en          1           1      100.00

Total Node Count     =         43
Toggled Node Count   =         43
Untoggled Node Count =          0

Toggle Coverage      =     100.00% (86 of 86 bins)
```

FUNCTIONAL COVERAGE

```
Covergroup Coverage:
    Covergroups                     1        na        na   100.00%
        Coverpoints/Crosses        22        na        na        na
            Covergroup Bins        78        78         0   100.00%
----------------------------------------------------------------------------------------------------
Covergroup                                          Metric       Goal       Bins    Status
```

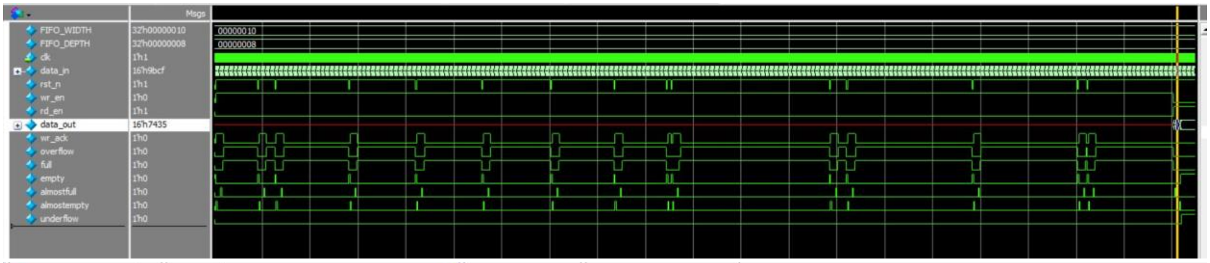
ASSERTIONS COVERAGE

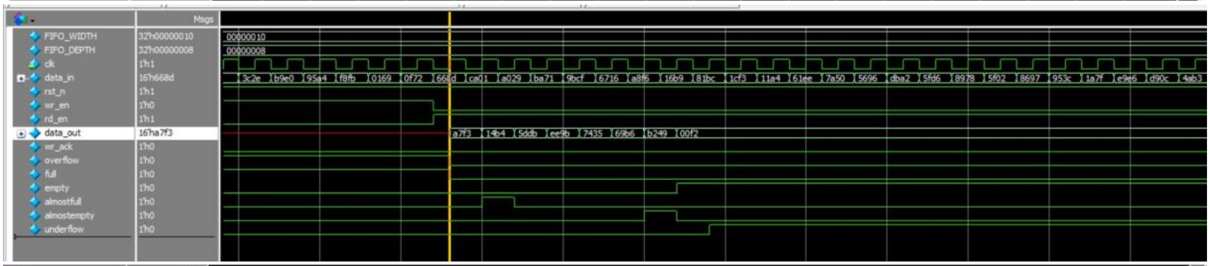
```
DIRECTIVE COVERAGE:
--------------------------------------------------------------------------------------------
Name                                    Design Design   Lang File(Line)      Hits Status
                                        Unit   UnitType
--------------------------------------------------------------------------------------------
/FIFO_top/dut/FIFO_SVA_IF/cover_count_valid
                                        FIFO_SVA Verilog  SVA  FIFO_SVA.sv(11) 2955 Covered
/FIFO_top/dut/FIFO_SVA_IF/cover_overflow_cond
                                        FIFO_SVA Verilog  SVA  FIFO_SVA.sv(19) 1125 Covered
/FIFO_top/dut/FIFO_SVA_IF/cover_underflow_cond
                                        FIFO_SVA Verilog  SVA  FIFO_SVA.sv(27)  973 Covered
/FIFO_top/dut/FIFO_SVA_IF/cover_full_cond
                                        FIFO_SVA Verilog  SVA  FIFO_SVA.sv(35) 2955 Covered
/FIFO_top/dut/FIFO_SVA_IF/cover_empty_cond
                                        FIFO_SVA Verilog  SVA  FIFO_SVA.sv(43) 2955 Covered
/FIFO_top/dut/FIFO_SVA_IF/cover_almostfull_cond
                                        FIFO_SVA Verilog  SVA  FIFO_SVA.sv(51) 2955 Covered
/FIFO_top/dut/FIFO_SVA_IF/cover_almostempty_cond
                                        FIFO_SVA Verilog  SVA  FIFO_SVA.sv(59) 2955 Covered
/FIFO_top/dut/FIFO_SVA_IF/cover_wr_ptr_valid
                                        FIFO_SVA Verilog  SVA  FIFO_SVA.sv(67) 3001 Covered
/FIFO_top/dut/FIFO_SVA_IF/cover_rd_ptr_valid
                                        FIFO_SVA Verilog  SVA  FIFO_SVA.sv(75) 3001 Covered

TOTAL DIRECTIVE COVERAGE: 100.00%  COVERS: 9

ASSERTION RESULTS:
```

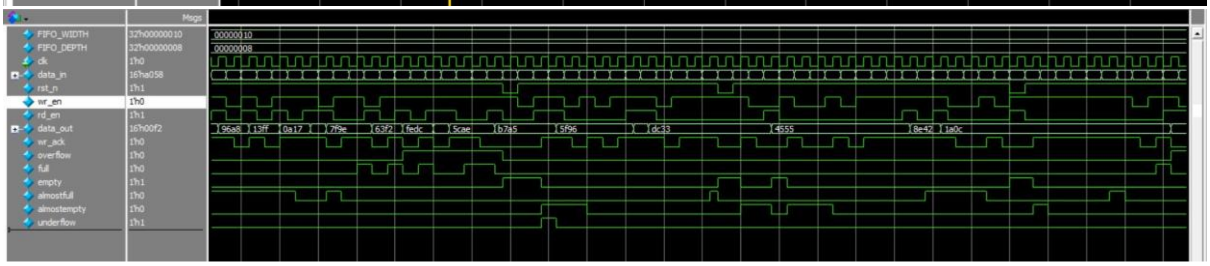
```
ASSERTION RESULTS:
---------------------------------------------------------------------
Name                  File(Line)                    Failure      Pass
                                                    Count        Count
---------------------------------------------------------------------
/FIFO_top/dut/FIFO_SVA_IF/assert_count_valid
                      FIFO_SVA.sv(10)                     0          1
/FIFO_top/dut/FIFO_SVA_IF/assert_overflow_cond
                      FIFO_SVA.sv(18)                     0          1
/FIFO_top/dut/FIFO_SVA_IF/assert_underflow_cond
                      FIFO_SVA.sv(26)                     0          1
/FIFO_top/dut/FIFO_SVA_IF/assert_full_cond
                      FIFO_SVA.sv(34)                     0          1
/FIFO_top/dut/FIFO_SVA_IF/assert_empty_cond
                      FIFO_SVA.sv(42)                     0          1
/FIFO_top/dut/FIFO_SVA_IF/assert_almostfull_cond
                      FIFO_SVA.sv(50)                     0          1
/FIFO_top/dut/FIFO_SVA_IF/assert_almostempty_cond
                      FIFO_SVA.sv(58)                     0          1
/FIFO_top/dut/FIFO_SVA_IF/assert_wr_ptr_valid
                      FIFO_SVA.sv(66)                     0          1
/FIFO_top/dut/FIFO_SVA_IF/assert_rd_ptr_valid
                      FIFO_SVA.sv(74)                     0          1
/FIFO_sequences/FIFO_main_wr/body/#ublk#84878499#66/immed__69
                      FIFO_sequences.sv(69)               0          1
```

SIMULATION RESULTS

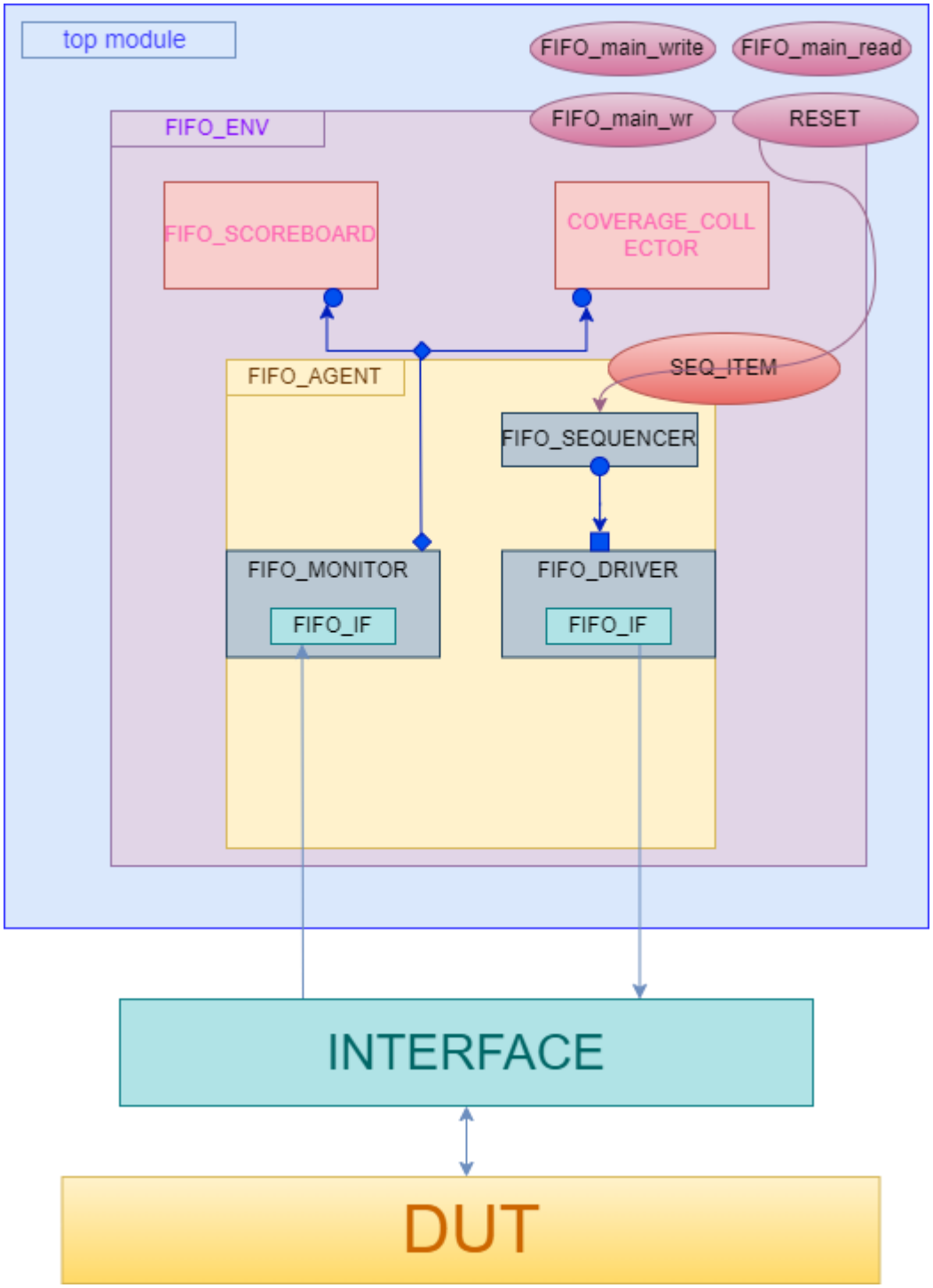
```
#   You are using a version of the UVM library that has been compiled
#   with `UVM_OBJECT_MUST_HAVE_CONSTRUCTOR undefined.
#   See http://www.eda.org/svdb/view.php?id=3770 for more details.
#
#       (Specify +UVM_NO_RELNOTES to turn off this notice)
#
# UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(277) @ 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(all)
# UVM_INFO @ 0: reporter [RNTST] Running test FIFO_test...
# UVM_INFO FIFO_TEST_pkg.sv(35) @ 0: uvm_test_top [run_phase] Resset asserted.
# ***************************************************************************
# * Questa UVM Transaction Recording Turned ON.                              *
# * recording_detail has been set.                                           *
# *  To turn off, set 'recording_detail' to off:                             *
# * uvm_config_db#(int)            ::set(null, "", "recording_detail", 0); *
# * uvm_config_db#(uvm_bitstream_t)::set(null, "", "recording_detail", 0); *
# ***************************************************************************
# UVM_INFO FIFO_TEST_pkg.sv(37) @ 2: uvm_test_top [run_phase] Resset deasserted.
# UVM_INFO FIFO_TEST_pkg.sv(39) @ 2: uvm_test_top [run_phase] stimulus generation started.
# UVM_INFO FIFO_TEST_pkg.sv(41) @ 2002: uvm_test_top [run_phase] Resset stimulus generation Ended.
# UVM_INFO FIFO_TEST_pkg.sv(44) @ 2002: uvm_test_top [run_phase] stimulus generation started.
# UVM_INFO FIFO_TEST_pkg.sv(46) @ 4002: uvm_test_top [run_phase] Resset stimulus generation Ended.
# UVM_INFO FIFO_TEST_pkg.sv(49) @ 4002: uvm_test_top [run_phase] stimulus generation started.
# UVM_INFO FIFO_TEST_pkg.sv(51) @ 6002: uvm_test_top [run_phase] Resset stimulus generation Ended.
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 6002: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO FIFO_scoreboard_pkg.sv(127) @ 6002: uvm_test_top.env.sb [report_phase] total successful transaction:3001
# UVM_INFO FIFO_scoreboard_pkg.sv(129) @ 6002: uvm_test_top.env.sb [reprt_phase] total failed transaction:0
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   14
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# [report_phase]     1
# [reprt_phase]     1
# [run_phase]     8
# ** Note: $finish    : C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 6002 ns  Iteration: 61  Instance: /FIFO_top
# Break in Task uvm_pkg/uvm_root::run_test at C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh line 430
```







Overview of the UVM Testbench for FIFO Verification



The Universal Verification Methodology (UVM) testbench designed for verifying a FIFO (First-In-First-Out) module is a comprehensive framework that systematically drives stimuli, monitors DUT (Design Under Test) behavior, and analyzes outputs to ensure correct functionality. Here's a summarized breakdown of how the UVM testbench operates:

1. Top Module Integration

The top module serves as the central hub, integrating the FIFO design with the UVM testbench environment. It instantiates the FIFO DUT and connects it to the verification components via a dedicated interface. This interface facilitates communication between the DUT and various UVM components such as drivers, monitors, and scoreboards.

2. Configuration Setup

A configuration package establishes the necessary settings and references for the testbench. It defines configuration objects that hold references to the virtual interface, ensuring that all verification components have consistent access to the DUT's signals. This setup is crucial for maintaining a unified environment where components can interact seamlessly.

3. Driving the Interface

The **FIFO_driver** component is responsible for generating and applying input transactions to the DUT. It fetches sequence items from the sequencer, which define specific operations like write or read commands, and drives these signals onto the DUT's interface. The driver ensures that the DUT receives a diverse set of stimuli, simulating real-world usage scenarios.

4. Generating Sequences

Sequences define the pattern and type of transactions sent to the DUT. Various sequences are implemented to cover different operational modes:

- **Reset Sequence:** Initializes the FIFO by asserting and deasserting the reset signal.
- **Write Sequence:** Generates a series of write operations to populate the FIFO.
- **Read Sequence:** Initiates read operations to retrieve data from the FIFO.
- **Mixed Read/Write Sequence:** Simultaneously performs read and write operations to test concurrent access scenarios.

These sequences utilize randomized and constrained transaction items to ensure comprehensive coverage of possible FIFO states and behaviors.

5. Monitoring DUT Behavior

The **FIFO_monitor** continuously observes the DUT's interface signals, capturing both inputs and outputs without altering them. It translates these signal states into transaction objects that are forwarded to analysis ports. This monitoring is essential for tracking the DUT's behavior in response to the driven stimuli and for collecting data for coverage and scoreboarding.

6. Coverage Collection

The **FIFO_coverage** component defines coverage groups that track the occurrence of various transaction combinations and states. By sampling transactions captured by the monitor, it ensures that all functional scenarios, such as different combinations of write and read operations and various FIFO status flags, are exercised during simulation. This comprehensive coverage analysis helps identify untested areas of the design.

7. Scoreboarding and Output Analysis

The **FIFO_scoreboard** plays a critical role in functional verification by comparing the DUT's outputs against a reference model. It receives transactions from the monitor, computes the expected behavior based on the input signals, and verifies that the DUT's outputs match the expected results. The scoreboard maintains counters for correct and erroneous transactions, providing detailed logs for any discrepancies found during simulation.

8. Assertions for Property Verification

SystemVerilog Assertions (SVAs) are embedded in the testbench to automatically check critical aspects of the FIFO's functionality. These assertions help ensure that the design operates correctly under specific conditions, such as checking for illegal states or validating correct signal transitions during simulation.defined depth or drops below zero.

9. Agent in the Environment

The **FIFO_agent** plays a crucial role in the environment by encapsulating the essential components needed for verification, including the sequencer, driver, and monitor. It is responsible for managing the creation and connection of these components during the build_phase and connect_phase. The **sequencer** generates transactions that are sent to the **driver**, which applies these stimuli to the DUT via the interface signals. Simultaneously, the **monitor** observes the DUT's outputs and forwards these transactions through an analysis port, enabling scoreboarding and coverage collection. The agent ensures that data flows seamlessly between the driver and monitor and that the stimuli and responses are properly tracked throughout the simulation. This coordination allows the testbench to apply stimuli and collect results efficiently, forming the backbone of the verification environment.

10. Environment and Agent Coordination

The **FIFO_Env** encapsulates all verification components, including agents, coverage collectors, and the scoreboard. It orchestrates the interactions between these components, ensuring that data flows correctly from drivers through monitors to scoreboards and coverage units. The environment manages the lifecycle of verification components and facilitates communication through analysis ports and exports.

11. Test Execution and Reporting

The **FIFO_test** class coordinates the overall verification process. It initializes the environment, configures components, and initiates the execution of various sequences. During the run phase, it raises objections to keep the simulation active until all activities are completed. After simulation, the testbench aggregates results from the scoreboard and coverage components, providing a summary of successful and failed transactions as well as coverage metrics. This reporting phase offers a clear overview of the FIFO's verification status, highlighting areas that passed or require further attention.

SystemVerilog Assertions for FIFO Functionality

Feature (Explanation)	Assertion
Count is always within valid limits	@(posedge FIFO_if.clk) disable iff(FIFO_if.rst_n==0) (FIFO_top.dut.count <= FIFO_DEPTH);
Overflow happens only when FIFO is full and write is enabled	@(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n)(FIFO_if.full && FIFO_if.wr_en) |=> FIFO_if.overflow == 1;
Underflow happens only when FIFO is empty and read is enabled	@(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n)(FIFO_if.empty && FIFO_if.rd_en) |=> FIFO_if.underflow == 1;
Full flag is set only when count equals FIFO depth	@(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n)(FIFO_if.full == (FIFO_top.dut.count == FIFO_DEPTH));
Empty flag is set only when count equals zero	@(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n)(FIFO_if.empty == (FIFO_top.dut.count == 0));
Almost full flag is set when count equals FIFO depth - 2	@(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n)(FIFO_if.almostfull == (FIFO_top.dut.count == FIFO_DEPTH-2));
Almost empty flag is set when count equals 1	@(posedge FIFO_if.clk) disable iff(!FIFO_if.rst_n)(FIFO_if.almostempty == (FIFO_top.dut.count == 1));
Write pointer always remains within FIFO depth	@(posedge FIFO_if.clk) (FIFO_top.dut.wr_ptr < FIFO_DEPTH);
Read pointer always remains within FIFO depth	@(posedge FIFO_if.clk) (FIFO_top.dut.rd_ptr < FIFO_DEPTH);