FIFO IMPLEMENTATION AND VERIFICATION USING THE SYSTEM VERILOG:-

```
module fifo(
 input clock, rd, wr,
 output full, empty,
 input [7:0] data_in,
 output reg [7:0] data_out,
 input rst);


 reg [7:0] mem [31:0];
 reg [4:0] wr_ptr;
 reg [4:0] rd_ptr;



always@(posedge clock)
 begin
  if (rst == 1'b1)
   begin
    data_out <= 0;
    rd_ptr <= 0;
    wr_ptr <= 0;
    for(int i =0; i < 32; i++) begin
     mem[i] <= 0;
    end
   end
  else
   begin
    if ((wr == 1'b1)  && (full == 1'b0))
     begin
```

```
          mem[wr_ptr] <= data_in;
          wr_ptr = wr_ptr + 1;
          end

        if((rd == 1'b1) && (empty == 1'b0))
          begin
          data_out <= mem[rd_ptr];
          rd_ptr <= rd_ptr + 1;
          end
      end
end


assign empty = ((wr_ptr - rd_ptr) == 0) ? 1'b1 : 1'b0;

assign full = ((wr_ptr - rd_ptr) == 31) ? 1'b1 : 1'b0;


endmodule
```

Interface

```
interface fifo_if;

  logic clock, rd, wr;
  logic full, empty;
  logic [7:0] data_in;
```

```
 logic [7:0] data_out;
 logic rst;

endinterface
```

Testbench Code

```
class transaction;

 rand bit rd ,wr;
 rand bit [7:0] data_in;
 bit full, empty;
 bit [7:0] data_out;

 constraint wr_rd {
   rd != wr;
   wr dist {0 :/ 50 , 1:/ 50};
   rd dist {0 :/ 50 , 1:/ 50};

 }


  constraint data_con {
 data_in > 1; data_in < 5;
 }


  function void display(input string tag);
```

```
    $display("[%0s] : WR : %0b\t RD:%0b\t DATAWR : %0d\t DATARD : %0d\t FULL : %0b\t EMPTY : %0b @ %0t", tag, wr, rd, data_in, data_out, full, empty,$time);
  endfunction

  function transaction copy();
    copy = new();
    copy.rd = this.rd;
    copy.wr = this.wr;
    copy.data_in = this.data_in;
    copy.data_out= this.data_out;
    copy.full = this.full;
    copy.empty = this.empty;
  endfunction

endclass

/*

module tb;

  transaction tr;

  initial begin
    tr = new();
    tr.display("TOP");
  end



endmodule
```

```
*/




class generator;

  transaction tr;
  mailbox #(transaction) mbx;

  int count = 0;

  event next;  ///know when to send next transaction
  event done;  ////conveys completion of requested no. of transaction


 function new(mailbox #(transaction) mbx);
    this.mbx = mbx;
    tr=new();
  endfunction;


  task run();

    repeat(count)
         begin
      assert(tr.randomize) else $error("Randomization failed");
      mbx.put(tr.copy);
      tr.display("GEN");
      @(next);
      end



    ->done;
  endtask



endclass

/*
 module tb;

   generator gen;
   mailbox #(transaction) mbx;




   initial begin
     mbx = new();

     gen = new(mbx);

     gen.count = 20;
     gen.run();



   end
 endmodule

 */



class driver;

  virtual fifo_if fif;

  mailbox #(transaction) mbx;

  transaction datac;

  event next;


   function new(mailbox #(transaction) mbx);
     this.mbx = mbx;
  endfunction;

 ////reset DUT
```

```
  task reset();
    fif.rst <= 1'b1;
    fif.rd <= 1'b0;
    fif.wr <= 1'b0;
    fif.data_in <= 0;
    repeat(5) @(posedge fif.clock);
    fif.rst <= 1'b0;
    $display("[DRV] : DUT Reset Done");
  endtask

  //////Applying RANDOM STIMULUS TO DUT
  task run();
    forever begin
      mbx.get(datac);

      datac.display("DRV");

      fif.rd <= datac.rd;
      fif.wr <= datac.wr;
      fif.data_in <= datac.data_in;
      repeat(2) @(posedge fif.clock);
      ->next;
    end
  endtask


endclass


/*
```

```
 module tb;

   generator gen;

   driver drv;

   event next;

   mailbox #(transaction) mbx;

   fifo_if fif();

   fifo dut (fif.clock, fif.rd, fif.wr,fif.full, fif.empty, fif.data_in, fif.data_out,
fif.rst);

   initial begin
     fif.clock <= 0;
   end

   always #10 fif.clock <= ~fif.clock;

   initial begin
     mbx = new();

     gen = new(mbx);

     gen.count = 20;

     drv = new(mbx);

     drv.fif = fif;
```

```
    gen.next = next;
    drv.next = next;


  end

  initial begin
    fork
      gen.run();
      drv.run();
    join
  end

  initial begin
    #800;
    $finish();
  end

  initial begin
    $dumpfile("dump.vcd");
    $dumpvars;
  end


endmodule

*/
```

```
class monitor;

  virtual fifo_if fif;

  mailbox #(transaction) mbx;

  transaction tr;




   function new(mailbox #(transaction) mbx);
     this.mbx = mbx;
  endfunction;


 task run();
   tr = new();

   forever begin
     repeat(2) @(posedge fif.clock);
     tr.wr = fif.wr;
     tr.rd = fif.rd;
     tr.data_in = fif.data_in;
     tr.data_out = fif.data_out;
     tr.full = fif.full;
     tr.empty = fif.empty;


     mbx.put(tr);
     tr.display("MON");

   end

  endtask



endclass

//////////////////////////////////////////////////////


class scoreboard;

   mailbox #(transaction) mbx;

   transaction tr;

   event next;

  bit [7:0] din[$];
  bit[7:0] temp;

   function new(mailbox #(transaction) mbx);
     this.mbx = mbx;
    endfunction;



  task run();
```

```
forever begin

  mbx.get(tr);

  tr.display("SCO");

  if(tr.wr == 1'b1)
    begin
    din.push_front(tr.data_in);
      $display("[SCO] : DATA STORED IN QUEUE :%0d", tr.data_in);
    end

  if(tr.rd == 1'b1)
    begin
      if(tr.empty == 1'b0) begin

        temp = din.pop_back();

        if(tr.data_out == temp)
          $display("[SCO] : DATA MATCH");
         else
           $error("[SCO] : DATA MISMATCH");
      end
      else
        begin
          $display("[SCO] : FIFO IS EMPTY");
        end


   end
```

```
    ->next;
  end
  endtask


endclass
////////////////////////////////////////


/*
module tb;


  monitor mon;
  scoreboard sco;

  mailbox #(transaction) mbx;

  event next;


   fifo_if fif();

   fifo dut (fif.clock, fif.rd, fif.wr,fif.full, fif.empty, fif.data_in, fif.data_out,
fif.rst);

    initial begin
      fif.clock <= 0;
    end
  always #10 fif.clock <= ~fif.clock;



initial begin
  mbx = new();

  mon = new(mbx);
  sco = new(mbx);

  mon.fif = fif;

  mon.next = next;
  sco.next = next;
end

initial begin
  fork
    mon.run();
    sco.run();
  join
end

    initial begin
    #200;
    $finish();
  end

  initial begin
    $dumpfile("dump.vcd");
    $dumpvars;
    end





endmodule

*/





class environment;

    generator gen;
    driver drv;

    monitor mon;
    scoreboard sco;

  mailbox #(transaction) gdmbx; ///generator + Driver

  mailbox #(transaction) msmbx; ///Monitor + Scoreboard

  event nextgs;


  virtual fifo_if fif;
```

```
function new(virtual fifo_if fif);



  gdmbx = new();
  gen = new(gdmbx);
  drv = new(gdmbx);




  msmbx = new();
  mon = new(msmbx);
  sco = new(msmbx);


  this.fif = fif;

  drv.fif = this.fif;
  mon.fif = this.fif;


  gen.next = nextgs;
  sco.next = nextgs;

endfunction




task pre_test();
  drv.reset();
```

```
 endtask

 task test();
 fork
  gen.run();
  drv.run();
  mon.run();
  sco.run();
 join_any

 endtask

 task post_test();
  wait(gen.done.triggered);
  $finish();
 endtask

 task run();
  pre_test();
  test();
  post_test();
 endtask



endclass
```

```
 module tb;



   fifo_if fif();
   fifo dut (fif.clock, fif.rd, fif.wr,fif.full, fif.empty, fif.data_in, fif.data_out,
fif.rst);

   initial begin
     fif.clock <= 0;
   end

   always #10 fif.clock <= ~fif.clock;

   environment env;



   initial begin
     env = new(fif);
     env.gen.count = 20;
     env.run();
   end


   initial begin
     $dumpfile("dump.vcd");
     $dumpvars;
   end
```

```
endmodule
```