DAY-87
#100DAYSRTL

"Aim":-Verification of Synchronous FIFO using System Verilog

"TestBench Codes":-

- **Design Code:-**

```
module FIFO(input clk, rst, wr, rd,
            input [7:0] din, output reg [7:0] dout,
            output empty, full);
  // Pointers for write and read operations
  reg [3:0] wptr = 0, rptr = 0;
  // Counter for tracking the number of elements in the FIFO
  reg [4:0] cnt = 0;
  // Memory array to store data
  reg [7:0] mem [15:0];
  always @(posedge clk)
    begin
      if (rst == 1'b1)
        begin
          // Reset the pointers and counter when the reset signal is asserted
          wptr <= 0;
          rptr <= 0;
          cnt  <= 0;
        end
      else if (wr && !full)
        begin
          // Write data to the FIFO if it's not full
          mem[wptr] <= din;
          wptr      <= wptr + 1;
          cnt       <= cnt + 1;
        end
      else if (rd && !empty)
        begin
          // Read data from the FIFO if it's not empty
          dout <= mem[rptr];
          rptr <= rptr + 1;
          cnt  <= cnt - 1;
        end
    end
  // Determine if the FIFO is empty or full
  assign empty = (cnt == 0) ? 1'b1 : 1'b0;
  assign full  = (cnt == 16) ? 1'b1 : 1'b0;
endmodule
```

- **Interface:-**

```
interface fifo_if;
  logic clock, rd, wr;          // Clock, read, and write signals
  logic full, empty;            // Flags indicating FIFO status
  logic [7:0] data_in;          // Data input
  logic [7:0] data_out;         // Data output
  logic rst;                    // Reset signal
endinterface
```

- **Transaction:-**

```
class transaction;
  rand bit oper;            // Randomized bit for operation control (1 or 0)
  bit rd, wr;               // Read and write control bits
  bit [7:0] data_in;        // 8-bit data input
  bit full, empty;          // Flags for full and empty status
  bit [7:0] data_out;       // 8-bit data output
  constraint oper_ctrl {
    oper dist {1 :/ 50 , 0 :/ 50}; }
endclass
```

- # Generator:-

```
class generator;
  transaction tr;           // Transaction object to generate and send
  mailbox #(transaction) mbx;  // Mailbox for communication
  int count = 0;            // Number of transactions to generate
  int i = 0;                // Iteration counter
  event next;               // Event to signal when to send the next transaction
  event done;               // Event to convey completion of requested number of transactions
  function new(mailbox #(transaction) mbx);
    this.mbx = mbx;
    tr = new();
  endfunction;
  task run();
    repeat (count) begin
      assert (tr.randomize) else $error("Randomization failed");
      i++;
      mbx.put(tr);
      $display("[GEN] : Oper : %0d iteration : %0d", tr.oper, i);
      @(next);
    end -> done;
  endtask
endclass
```

- # Driver:-

```
class driver;
  virtual fifo_if fif;     // Virtual interface to the FIFO
  mailbox #(transaction) mbx;  // Mailbox for communication
  transaction datac;       // Transaction object for communication
  function new(mailbox #(transaction) mbx);
    this.mbx = mbx;
  endfunction;
  // Reset the 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");
    $display("------------------------------------------");
  endtask
  // Write data to the FIFO
  task write();
    @(posedge fif.clock);
    fif.rst <= 1'b0;
    fif.rd <= 1'b0;
    fif.wr <= 1'b1;
    fif.data_in <= $urandom_range(1, 10);
    @(posedge fif.clock);
    fif.wr <= 1'b0;
    $display("[DRV] : DATA WRITE  data : %0d", fif.data_in);
    @(posedge fif.clock);
  endtask
  // Read data from the FIFO
  task read();
    @(posedge fif.clock);
    fif.rst <= 1'b0;
    fif.rd <= 1'b1;
    fif.wr <= 1'b0;
    @(posedge fif.clock);
    fif.rd <= 1'b0;
    $display("[DRV] : DATA READ");
    @(posedge fif.clock);
  endtask
  // Apply random stimulus to the DUT
  task run();
    forever begin
      mbx.get(datac);
      if (datac.oper == 1'b1)
        write();
      else
        read();
    end
  endtask
endclass
```

- # Monitor:-

```
class monitor;
  virtual fifo_if fif;     // Virtual interface to the FIFO
  mailbox #(transaction) mbx;  // Mailbox for communication
  transaction tr;          // Transaction object for monitoring
  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.full = fif.full;
      tr.empty = fif.empty;
      @(posedge fif.clock);
      tr.data_out = fif.data_out;
      mbx.put(tr);
      $display("[MON] : Wr:%0d rd:%0d din:%0d dout:%0d full:%0d empty:%0d", tr.wr, tr.rd, tr.data_in, tr.data_out, tr.full, tr.empty);
    end
  endtask
endclass
```

- # Scoreboard:-

```
class scoreboard;
  mailbox #(transaction) mbx;  // Mailbox for communication
  transaction tr;          // Transaction object for monitoring
  event next;
  bit [7:0] din[$];       // Array to store written data
  bit [7:0] temp;         // Temporary data storage
  int err = 0;            // Error count
  function new(mailbox #(transaction) mbx);
    this.mbx = mbx;
  endfunction;
  task run();
    forever begin
      mbx.get(tr);
      $display("[SCO] : Wr:%0d rd:%0d din:%0d dout:%0d full:%0d empty:%0d", tr.wr, tr.rd, tr.data_in, tr.data_out, tr.full, tr.empty);
      if (tr.wr == 1'b1) begin
        if (tr.full == 1'b0) begin
          din.push_front(tr.data_in);
          $display("[SCO] : DATA STORED IN QUEUE :%0d", tr.data_in);
        end
        else begin
          $display("[SCO] : FIFO is full");
        end
        $display("-----------------------------------------");
      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 begin
            $error("[SCO] : DATA MISMATCH");
            err++;
          end
        end
        else begin
          $display("[SCO] : FIFO IS EMPTY");
        end
        $display("-----------------------------------------");
      end
      -> next;
    end
  endtask
endclass
```

- # Environment:-

```
class environment;
  generator gen;
  driver drv;
  monitor mon;
  scoreboard sco;
  mailbox #(transaction) gdmbx;  // Generator + Driver mailbox
  mailbox #(transaction) msmbx;  // Monitor + Scoreboard mailbox
  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);
    $display("-----------------------------------------");
    $display("Error Count :%0d", sco.err);
    $display("-----------------------------------------");
    $finish();
  endtask
  task run();
    pre_test();
    test();
    post_test();
  endtask
endclass
```

- # TB_Top:-

```
module tb;
  fifo_if fif();
  FIFO dut (fif.clock, fif.rst, fif.wr, fif.rd, fif.data_in, fif.data_out, fif.empty, fif.full);
  initial begin
    fif.clock <= 0;
  end
  always #10 fif.clock <= ~fif.clock;
  environment env;
  initial begin
    env = new(fif);
    env.gen.count = 10;
    env.run();
  end
  initial begin
    $dumpfile("dump.vcd");
    $dumpvars;
  end
endmodule
```