DAY-86
#100DAYSRTL

"Aim":-Verification of D FlipFlop using Layered Testbench

"TestBench Codes":-

- **Design Code:-**

```
module dff (dff_if vif);
  always @(posedge vif.clk)
    begin
      if (vif.rst == 1'b1)
        vif.dout <= 1'b0;
      else
        vif.dout <= vif.din;
    end
endmodule
```

- **Interface:-**

```
interface dff_if;
  logic clk;   // Clock signal
  logic rst;   // Reset signal
  logic din;   // Data input
  logic dout;  // Data output
endinterface
```

- **Transaction:-**

```
class transaction;
  rand bit din;     // Define a random input bit "din"
  bit dout;         // Define an output bit "dout"
  function transaction copy();
    copy = new();   // Create a new transaction object
    copy.din = this.din;  // Copy the input value
    copy.dout = this.dout;  // Copy the output value
  endfunction
  function void display(input string tag);
    $display("[%0s] : DIN : %0b DOUT : %0b", tag, din, dout);
  endfunction
endclass
```

- **Generator:-**

```
class generator;
  transaction tr;  // Define a transaction object
  mailbox #(transaction) mbx;      // Create a mailbox to send data to the driver
  mailbox #(transaction) mbxref;   // Create a mailbox to send data to the scoreboard
  event sconext; // Event to sense the completion of scoreboard work
  event done;    // Event to trigger when the requested number of stimuli is applied
  int count;     // Stimulus count
  function new(mailbox #(transaction) mbx, mailbox #(transaction) mbxref);
    this.mbx = mbx;  // Initialize the mailbox for the driver
    this.mbxref = mbxref; // Initialize the mailbox for the scoreboard
    tr = new(); // Create a new transaction object
  endfunction
  task run();
    repeat(count) begin
      assert(tr.randomize) else $error("[GEN] : RANDOMIZATION FAILED");
      mbx.put(tr.copy); // Put a copy of the transaction into the driver mailbox
      mbxref.put(tr.copy); // Put a copy of the transaction into the scoreboard mailbox
      tr.display("GEN"); // Display transaction information
      @(sconext); // Wait for the scoreboard's completion signal
    end
    ->done; // Trigger "done" event when all stimuli are applied
  endtask
endclass
```

- **Driver:-**

```
class driver;
  transaction tr; // Define a transaction object
  mailbox #(transaction) mbx; // Create a mailbox to receive data from the generator
  virtual dff_if vif; // Virtual interface for DUT
  function new(mailbox #(transaction) mbx);
    this.mbx = mbx; // Initialize the mailbox for receiving data
  endfunction
  task reset();
    vif.rst <= 1'b1; // Assert reset signal
    repeat(5) @(posedge vif.clk); // Wait for 5 clock cycles
    vif.rst <= 1'b0; // Deassert reset signal
    @(posedge vif.clk); // Wait for one more clock cycle
    $display("[DRV] : RESET DONE"); // Display reset completion message
  endtask
  task run();
    forever begin
      mbx.get(tr); // Get a transaction from the generator
      vif.din <= tr.din; // Set DUT input from the transaction
      @(posedge vif.clk); // Wait for the rising edge of the clock
      tr.display("DRV"); // Display transaction information
      vif.din <= 1'b0; // Set DUT input to 0
      @(posedge vif.clk); // Wait for the rising edge of the clock
    end
  endtask
endclass
```

- **Monitor:-**

```
class monitor;
  transaction tr; // Define a transaction object
  mailbox #(transaction) mbx; // Create a mailbox to send data to the scoreboard
  virtual dff_if vif; // Virtual interface for DUT
  function new(mailbox #(transaction) mbx);
    this.mbx = mbx; // Initialize the mailbox for sending data to the scoreboard
  endfunction
  task run();
    tr = new(); // Create a new transaction
    forever begin
      repeat(2) @(posedge vif.clk); // Wait for two rising edges of the clock
      tr.dout = vif.dout; // Capture DUT output
      mbx.put(tr); // Send the captured data to the scoreboard
      tr.display("MON"); // Display transaction information
    end
  endtask
endclass
```

- **Scoreboard:-**

```
class scoreboard;
  transaction tr; // Define a transaction object
  transaction trref; // Define a reference transaction object for comparison
  mailbox #(transaction) mbx; // Create a mailbox to receive data from the driver
  mailbox #(transaction) mbxref; // Create a mailbox to receive reference data from the generator
  event sconext; // Event to signal completion of scoreboard work
  function new(mailbox #(transaction) mbx, mailbox #(transaction) mbxref);
    this.mbx = mbx; // Initialize the mailbox for receiving data from the driver
    this.mbxref = mbxref; // Initialize the mailbox for receiving reference data from the generator
  endfunction
  task run();
    forever begin
      mbx.get(tr); // Get a transaction from the driver
      mbxref.get(trref); // Get a reference transaction from the generator
      tr.display("SCO"); // Display the driver's transaction information
      trref.display("REF"); // Display the reference transaction information
      if (tr.dout == trref.din)
        $display("[SCO] : DATA MATCHED"); // Compare data and display the result
      else
        $display("[SCO] : DATA MISMATCHED");
      $display("-------------------------------------------------");
      ->sconext; // Signal completion of scoreboard work
    end
  endtask
endclass
```

- **Environment:-**

```
class environment;
  generator gen; // Generator instance
  driver drv; // Driver instance
  monitor mon; // Monitor instance
  scoreboard sco; // Scoreboard instance
  event next; // Event to signal communication between generator and scoreboard
  mailbox #(transaction) gdmbx; // Mailbox for communication between generator and driver
  mailbox #(transaction) msmbx; // Mailbox for communication between monitor and scoreboard
  mailbox #(transaction) mbxref; // Mailbox for communication between generator and scoreboard
  virtual dff_if vif; // Virtual interface for DUT
  function new(virtual dff_if vif);
    gdmbx = new(); // Create a mailbox for generator-driver communication
    mbxref = new(); // Create a mailbox for generator-scoreboard reference data
    gen = new(gdmbx, mbxref); // Initialize the generator
    drv = new(gdmbx); // Initialize the driver
    msmbx = new(); // Create a mailbox for monitor-scoreboard communication
    mon = new(msmbx); // Initialize the monitor
    sco = new(msmbx, mbxref); // Initialize the scoreboard
    this.vif = vif; // Set the virtual interface for DUT
    drv.vif = this.vif; // Connect the virtual interface to the driver
    mon.vif = this.vif; // Connect the virtual interface to the monitor
    gen.sconext = next; // Set the communication event between generator and scoreboard
    sco.sconext = next; // Set the communication event between scoreboard and generator
  endfunction
  task pre_test();
    drv.reset(); // Perform the driver reset
  endtask
  task test();
    fork
      gen.run(); // Start generator
      drv.run(); // Start driver
      mon.run(); // Start monitor
      sco.run(); // Start scoreboard
    join_any
  endtask
  task post_test();
    wait(gen.done.triggered); // Wait for generator to complete
    $finish(); // Finish simulation
  endtask
  task run();
    pre_test(); // Run pre-test setup
    test(); // Run the test
    post_test(); // Run post-test cleanup
  endtask
endclass
```

- **Tb_top:-**

```
module tb;
  dff_if vif(); // Create DUT interface
  dff dut(vif); // Instantiate DUT
  initial begin
    vif.clk <= 0; // Initialize clock signal
  end
  always #10 vif.clk <= ~vif.clk; // Toggle the clock every 10 time units
  environment env; // Create environment instance
  initial begin
    env = new(vif); // Initialize the environment with the DUT interface
    env.gen.count = 30; // Set the generator's stimulus count
    env.run(); // Run the environment
  end
  initial begin
    $dumpfile("dump.vcd"); // Specify the VCD dump file
    $dumpvars; // Dump all variables
  end
endmodule
```

“<u>Result</u>”:-

```
[DRV] : RESET DONE
[GEN] : DIN : 1 DOUT : 0
[DRV] : DIN : 1 DOUT : 0
[MON] : DIN : 0 DOUT : 1
[SCO] : DIN : 0 DOUT : 1
[REF] : DIN : 1 DOUT : 0
[SCO] : DATA MATCHED
--------------------------------------------------
[GEN] : DIN : 0 DOUT : 0
[DRV] : DIN : 0 DOUT : 0
[MON] : DIN : 0 DOUT : 0
[SCO] : DIN : 0 DOUT : 0
[REF] : DIN : 0 DOUT : 0
[SCO] : DATA MATCHED
--------------------------------------------------
[GEN] : DIN : 0 DOUT : 0
[DRV] : DIN : 0 DOUT : 0
[MON] : DIN : 0 DOUT : 0
[SCO] : DIN : 0 DOUT : 0
[REF] : DIN : 0 DOUT : 0
[SCO] : DATA MATCHED
--------------------------------------------------
[GEN] : DIN : 0 DOUT : 0
[DRV] : DIN : 0 DOUT : 0
[MON] : DIN : 0 DOUT : 0
[SCO] : DIN : 0 DOUT : 0
[REF] : DIN : 0 DOUT : 0
[SCO] : DATA MATCHED
--------------------------------------------------
[GEN] : DIN : 1 DOUT : 0
[DRV] : DIN : 1 DOUT : 0
[MON] : DIN : 0 DOUT : 1
[SCO] : DIN : 0 DOUT : 1
[REF] : DIN : 1 DOUT : 0
[SCO] : DATA MATCHED
--------------------------------------------------
[GEN] : DIN : 0 DOUT : 0
[DRV] : DIN : 0 DOUT : 0
[MON] : DIN : 0 DOUT : 0
[SCO] : DIN : 0 DOUT : 0
[REF] : DIN : 0 DOUT : 0
[SCO] : DATA MATCHED
--------------------------------------------------
[GEN] : DIN : 0 DOUT : 0
[DRV] : DIN : 0 DOUT : 0
[MON] : DIN : 0 DOUT : 0
[SCO] : DIN : 0 DOUT : 0
[REF] : DIN : 0 DOUT : 0
[SCO] : DATA MATCHED
--------------------------------------------------
[GEN] : DIN : 1 DOUT : 0
[DRV] : DIN : 1 DOUT : 0
[MON] : DIN : 0 DOUT : 1
[SCO] : DIN : 0 DOUT : 1
[REF] : DIN : 1 DOUT : 0
[SCO] : DATA MATCHED
--------------------------------------------------
[GEN] : DIN : 0 DOUT : 0
[DRV] : DIN : 0 DOUT : 0
[MON] : DIN : 0 DOUT : 0
[SCO] : DIN : 0 DOUT : 0
[REF] : DIN : 0 DOUT : 0
[SCO] : DATA MATCHED
--------------------------------------------------
[GEN] : DIN : 1 DOUT : 0
[DRV] : DIN : 1 DOUT : 0
[MON] : DIN : 0 DOUT : 1
[SCO] : DIN : 0 DOUT : 1
[REF] : DIN : 1 DOUT : 0
[SCO] : DATA MATCHED
```