**Nirma University Institute of Technology**

**Minor in VLSI Design**

**Semester – VI (EVEN 2024-25)**

****

**3EC608IC24 – Essentials of VLSI**

**Design Test**

**SPECIAL ASSIGNMENT**

**22BCE281, 22BCE296**

**RTL to GDS - Dual Port RAM**

# 

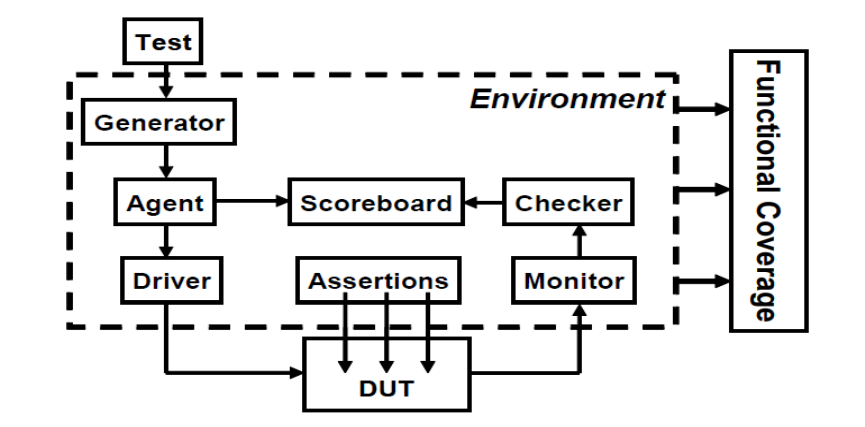
# Introductuion

Sequential circuits play a crucial role in digital design, as they allow for memory elements that store and manipulate data based on clock cycles. Among these, the D Flip-Flop (DFF) is one of the fundamental building blocks, widely used in registers, shift registers, and memory elements. The D Flip-Flop stores the input value on the rising edge of the clock and maintains its state until the next clock cycle. This characteristic makes it essential in synchronous digital systems where state retention is critical for proper functionality.

To ensure the correct operation of the D Flip-Flop in real-world applications, its functionality must be verified using a structured testbench. Verification is key in modern digital design, where functional correctness must be ensured before fabrication. SystemVerilog, an extension of Verilog, provides advanced constructs for design verification, including object-oriented programming (OOP) features, constrained random testing, assertions, and functional coverage. These features make SystemVerilog a preferred choice for writing robust and reusable verification environments.

# Testbench Architecture

A **layered testbench architecture** is a structured approach to verification that breaks down the testbench into different functional components, each responsible for a specific task. This modular structure improves reusability, maintainability, and scalability in verification environments. Inspired by **Universal Verification Methodology (UVM)**, the layered approach ensures a **clear separation of stimulus generation, DUT interaction, and result checking**, making it easier to develop and debug verification environments. It consistes of different layers like classes as follows



# Methodology/Code

// --------------------------------------

// D Flip-Flop Interface

// --------------------------------------

interface dff\_if;

logic clk, rst; // Clock and Reset signals

logic d, q; // Input D and Output Q

endinterface

// --------------------------------------

// D Flip-Flop Design Under Test (DUT)

// --------------------------------------

module dff(input logic d, clk, rst, output logic q);

always\_ff @(posedge clk or posedge rst) begin

if (rst)

q <= 0; // Reset behavior

else

q <= d; // D Flip-Flop functionality

end

endmodule

// --------------------------------------

// Transaction Class: Defines data format

// --------------------------------------

class transaction;

bit d, rst; // Randomized input

bit q; // Observed output

// Display transaction data

function void display(string tag);

$display("[%s] d: %0b q: %0b rst: %0b", tag, d, q, rst);

endfunction

endclass

// --------------------------------------

// Generator: Creates Stimulus

// --------------------------------------

class generator;

virtual dff\_if vif;

mailbox #(transaction) gen2drv; // To Driver

mailbox #(transaction) gen2sb; // To Scoreboard

int count; // Number of transactions to generate

function new(mailbox #(transaction) gen2drv, mailbox #(transaction) gen2sb);

this.gen2drv = gen2drv;

this.gen2sb = gen2sb;

endfunction

task run();

transaction tr;

repeat (count) begin

tr = new();

tr.d = $random % 2; // Generates 0 or 1

tr.rst = $random % 2; // Generates 0 or 1

gen2drv.put(tr); // Send to Driver

gen2sb.put(tr); // Send to Scoreboard

tr.display("GEN"); // Display generated transaction

end

endtask

endclass

// --------------------------------------

// Driver: Drives Inputs to DUT

// --------------------------------------

class driver;

virtual dff\_if vif;

mailbox #(transaction) gen2drv;

function new(mailbox #(transaction) gen2drv, virtual dff\_if vif);

this.gen2drv = gen2drv;

this.vif = vif;

endfunction

// Apply reset sequence

task reset();

vif.rst <= 1;

repeat (5) @(posedge vif.clk);

vif.rst <= 0;

$display("[DRV] Reset applied");

endtask

// Drive the DUT with input transactions

task run();

transaction tr;

forever begin

gen2drv.get(tr); // Receive from Generator

vif.d <= tr.d; // Apply input

vif.rst <= tr.rst; // Apply reset condition

@(posedge vif.clk); // Wait for clock edge

end

endtask

endclass

// --------------------------------------

// Monitor: Observes DUT Output

// --------------------------------------

class monitor;

virtual dff\_if vif;

mailbox #(transaction) mon2sb; // To Scoreboard

function new(mailbox #(transaction) mon2sb, virtual dff\_if vif);

this.mon2sb = mon2sb;

this.vif = vif;

endfunction

// Monitor DUT outputs and forward to scoreboard

task run();

transaction tr;

forever begin

@(posedge vif.clk); // Sync with clock

#5; // Delay for stable output

tr = new();

tr.q = vif.q;

tr.d = vif.d;

mon2sb.put(tr); // Send observed output

tr.display("MON");

end

endtask

endclass

// --------------------------------------

// Scoreboard: Checks DUT Correctness

// --------------------------------------

class scoreboard;

mailbox #(transaction) gen2sb; // From Generator

mailbox #(transaction) mon2sb; // From Monitor

function new(mailbox #(transaction) gen2sb, mailbox #(transaction) mon2sb);

this.gen2sb = gen2sb;

this.mon2sb = mon2sb;

endfunction

// Compare expected vs actual results

task run();

transaction gen\_tr, mon\_tr;

forever begin

gen2sb.get(gen\_tr); // Get expected value

mon2sb.get(mon\_tr); // Get observed value

gen\_tr.display("SCO\_GEN");

mon\_tr.display("SCO\_MON");

if (gen\_tr.d === mon\_tr.q || (mon\_tr.q === 0 && gen\_tr.rst === 1))

$display("[SCO] Match");

else

$display("[SCO] Mismatch");

end

endtask

endclass

// --------------------------------------

// Environment: Instantiates and Runs All Components

// --------------------------------------

class environment;

generator gen;

driver drv;

monitor mon;

scoreboard sco;

// Mailboxes for communication

mailbox #(transaction) gen2drv;

mailbox #(transaction) gen2sb;

mailbox #(transaction) mon2sb;

virtual dff\_if vif;

function new(virtual dff\_if vif);

this.vif = vif;

// Create mailboxes

gen2drv = new();

gen2sb = new();

mon2sb = new();

// Instantiate components

gen = new(gen2drv, gen2sb);

drv = new(gen2drv, vif);

mon = new(mon2sb, vif);

sco = new(gen2sb, mon2sb);

endfunction

// Run the full simulation environment

task run();

drv.reset(); // Apply reset once at start

fork

gen.run(); // Generate stimulus

drv.run(); // Drive DUT

mon.run(); // Monitor output

sco.run(); // Compare results

join\_none

endtask

endclass

// --------------------------------------

// Top-Level Testbench Module

// --------------------------------------

module tb;

dff\_if vif(); // Instantiate interface

environment env; // Environment instance

// Clock generation

initial vif.clk = 0;

always #10 vif.clk = ~vif.clk;

dff dut(.d(vif.d), .q(vif.q), .clk(vif.clk), .rst(vif.rst)); // Instantiate DUT

// Start the simulation

initial begin

env = new(vif); // Create environment

env.gen.count = 20; // Set number of test cases

env.run(); // Start environment

end

// Dump waveform for EPWave

initial begin

$dumpfile("dump.vcd"); // Specify VCD file

$dumpvars(0, tb); // Dump all signals in tb module

//#500 $finish;

end

endmodule

# Output

#[SCO] Match

# [MON] d: 0 q: 1 rst: 0

# [SCO\_GEN] d: 1 q: 0 rst: 0

# [SCO\_MON] d: 0 q: 1 rst: 0

# [SCO] Match

# [MON] d: 0 q: 0 rst: 0

# [SCO\_GEN] d: 0 q: 0 rst: 0

# [SCO\_MON] d: 0 q: 0 rst: 0

# [SCO] Match

# [MON] d: 0 q: 0 rst: 0

# [SCO\_GEN] d: 0 q: 0 rst: 1

# [SCO\_MON] d: 0 q: 0 rst: 0

# [SCO] Match

# [MON] d: 1 q: 0 rst: 0

# [SCO\_GEN] d: 0 q: 0 rst: 1

# [SCO\_MON] d: 1 q: 0 rst: 0

# [SCO] Match

# [MON] d: 1 q: 0 rst: 0

# [SCO\_GEN] d: 1 q: 0 rst: 1

# [SCO\_MON] d: 1 q: 0 rst: 0

# [SCO] Match

