Verification of Digital Systems
Layered Testbench for D Flip-Flop

Kushal M – PES2UG23EC072
Pranav M – PES2UG23EC076

December 2025

Contents

1 Introduction

This project involves implementing and verifying a D Flip-Flop using a layered testbench. The layered approach separates stimulus generation, DUT interfacing, and checking logic for clarity and reusability. All code segments are available here.

2 RTL Code

Here is the RTL Implementation of the D Flip-Flop:

```
module d_ff(input logic d, clk, output logic q);

always@(posedge clk)
q <= d;

endmodule
```

3 Layered Testbench

3.1 Transaction Class

Defines a transaction object that carries stimulus (d, clk) and the sampled output (q), along with a helper display function.

```
class transaction;

  rand bit d;
  bit clk;

  bit q;

  function void display(string name);
    $display("-------------------------");
    $display(" %s ", name);
    $display("-------------------------");
    $display("d = %0d, clk = %0d", d, clk);
    $display("q = %0d", q);
    $display("-------------------------");
  endfunction

endclass
```

3.2 Generator

Generates randomized transactions, displays them, and sends each one to the driver through a mailbox.

```
class generator;
  mailbox gen2drive;
  transaction trans;

  function new(mailbox gen2drive);
    this.gen2drive=gen2drive;
  endfunction

  task main();
    repeat(3)
      begin
        trans=new();
        trans.randomize();
        trans.display("generator");
        gen2drive.put(trans);
      end
  endtask
endclass
```

3.3 Driver

Drives each received transaction onto the interface signals and applies the clocking needed to stimulate the DUT.

```
class driver;
  mailbox gen2drive;
  virtual intf vif;

  function new(virtual intf vif,mailbox gen2drive);
    this.vif = vif;
    this.gen2drive = gen2drive;
  endfunction

  task main;
    repeat(3)
      begin
        transaction trans;
        gen2drive.get(trans);

        vif.d <= trans.d;
```

```
        vif.clk <=0;
        #5;
        vif.clk<=1;
        #5;
        trans.display("Driver");
      end
  endtask

endclass
```

3.4 Interface

Defines the shared interface carrying the signals connecting the testbench and DUT.

```
interface intf();
  logic d;
  logic clk;
  logic q;
endinterface
```

3.5 Monitor

Samples DUT signals on each clock edge, packages them into a transaction, and forwards them to the scoreboard.

```
class monitor;
  virtual intf vif;
  mailbox mon2sb;

  function new(virtual intf vif, mailbox mon2sb);
    this.vif     = vif;
    this.mon2sb  = mon2sb;
  endfunction

 task main;
    repeat(3) begin
      transaction trans;

      @(posedge vif.clk);
      #1;

      trans = new();
      trans.d   = vif.d;
      trans.clk = vif.clk;
      trans.q   = vif.q;
      mon2sb.put(trans);
```

```
      trans.display("monitor");
    end
  endtask

endclass
```

3.6 Scoreboard

Receives transactions from the monitor and checks that q matches d.

```
class scoreboard;
  mailbox mon2sb;

  function new(mailbox mon2sb);
    this.mon2sb= mon2sb;
  endfunction

task main;
  repeat(3
) begin
    transaction trans;
    mon2sb.get(trans);

    if (trans.q == trans.d)
      $display("result is as expected");
    else
      $error("Wrong Result");

    trans.display("scoreboard");
  end
endtask

endclass
```

3.7 Environment

Instantiates all testbench components, connects them via mailboxes, and initializes their execution.

```
‘include "transaction.sv"
‘include "generator.sv"
‘include "driver.sv"
‘include "monitor"
‘include "scoreboard"
class environment;
```

```
  generator gen;
  driver drv;
  monitor mon;
  scoreboard sb;

  mailbox gen2drive;
  mailbox mon2sb;

  virtual intf vif;

  function new(virtual intf vif);
    this.vif = vif;

    gen2drive = new();
    mon2sb    = new();

    gen = new(gen2drive);
    drv = new(vif, gen2drive);
    mon = new(vif, mon2sb);
    sb  = new(mon2sb);
  endfunction

  task run;
    fork
      gen.main();
      drv.main();
      mon.main();
      sb.main();
    join
  endtask

endclass
```

3.8 Test Block

Creates the environment with the DUT interface and runs the layered testbench.

```
‘include "environment.sv"
program test(intf i_intf);
  environment env;

  initial begin
    env = new(i_intf);
    env.run();
  end
```

```
endprogram
```

3.9 Testbench

Instantiates the DUT and interface, connects them, and runs the test program while enabling waveform dumping.

```
`include "interface.sv"
`include "test"
module testbench;

  intf i_intf();

  d_ff dut(
    .d(i_intf.d),
    .clk(i_intf.clk),
    .q(i_intf.q)
  );

  test t1(i_intf);

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

endmodule
```

4 Simulation Outputs

„

```
#  generator
# -------------------------
# d = 0, clk = 0
# q = 0
# -------------------------
# -------------------------
#  monitor
# -------------------------
# d = 0, clk = 1
# q = 0
# -------------------------
# result is as expected
# -------------------------
#  scoreboard
# -------------------------
# d = 0, clk = 1
# q = 0
```

"

Figure 1: Simulation waveform output 1.

```
# -------------------------
#  monitor
# -------------------------
# d = 1, clk = 1
# q = 1
# -------------------------
# result is as expected
# -------------------------
#  scoreboard
# -------------------------
# d = 1, clk = 1
# q = 1
# -------------------------
# -------------------------
```

Figure 2: Simulation waveform output 2.

5 Conclusion

In this project, a layered testbench for a D Flip-Flop was developed, comprising generator, driver, monitor, and scoreboard components. Each component was verified to work correctly, and simulation results confirmed that the D Flip-Flop output matched expected behavior.