VERIFICATION OF DIGITAL DESIGN

SHIFT REGISTER

BINITH M R | PES2UG22EC034
SANTOSH HANAMAPPA MOKASHI | PES2UG22EC118

INTRODUCTION

Firstly, what are shift register?

A shift register is a digital circuit that stores and manipulates binary data by shifting bits from flip-flop which are connected in a series of chain, where the output of each flip-flop is connected to the input of the next. When the clock signal ticks, the output of one flip-flop shits to the next.

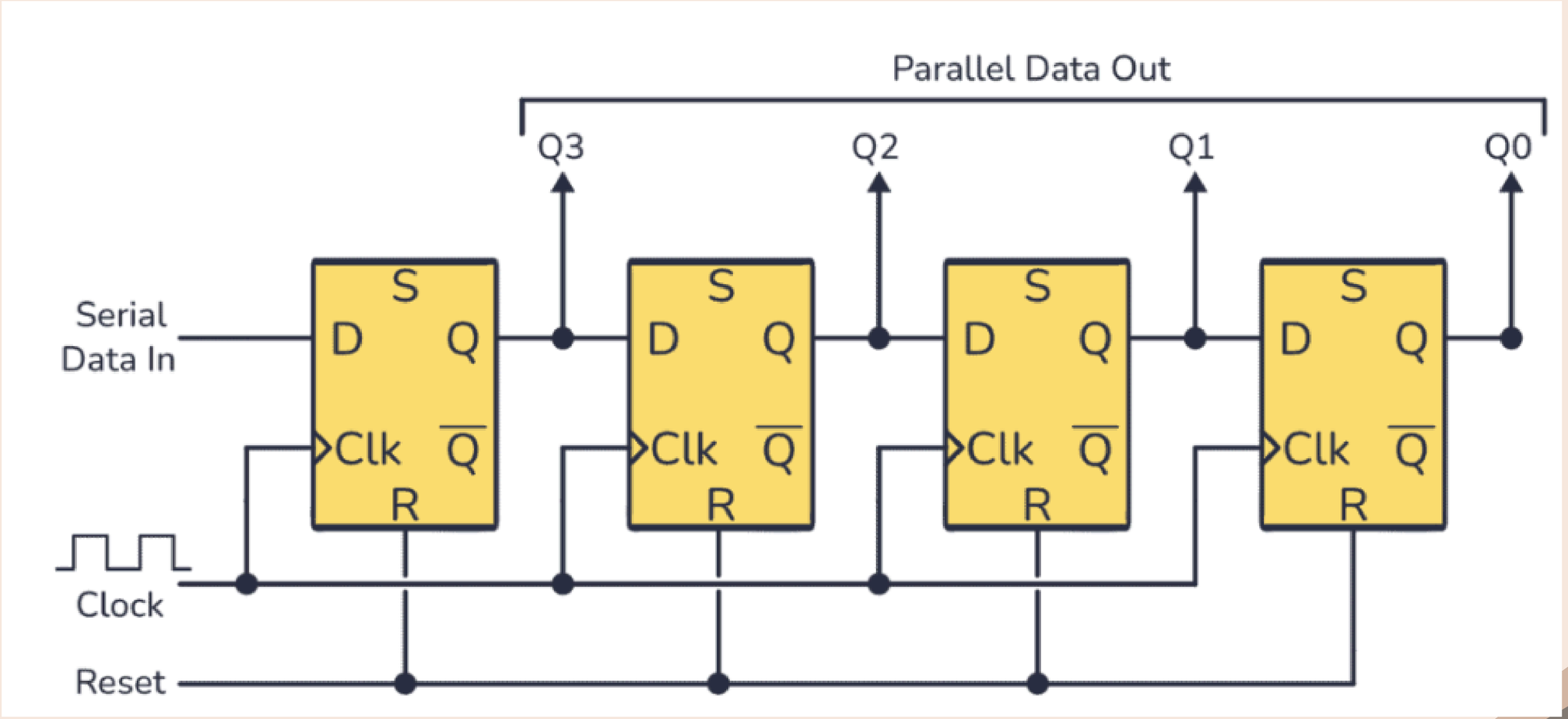
There are 4 types of shift registers:

i) Serial in serial out(SISO)
ii) Serial in parallel out(SIPO)
iii) Parallel in serial out(PISO)
iv) Parallel in parallel out(PIPO)

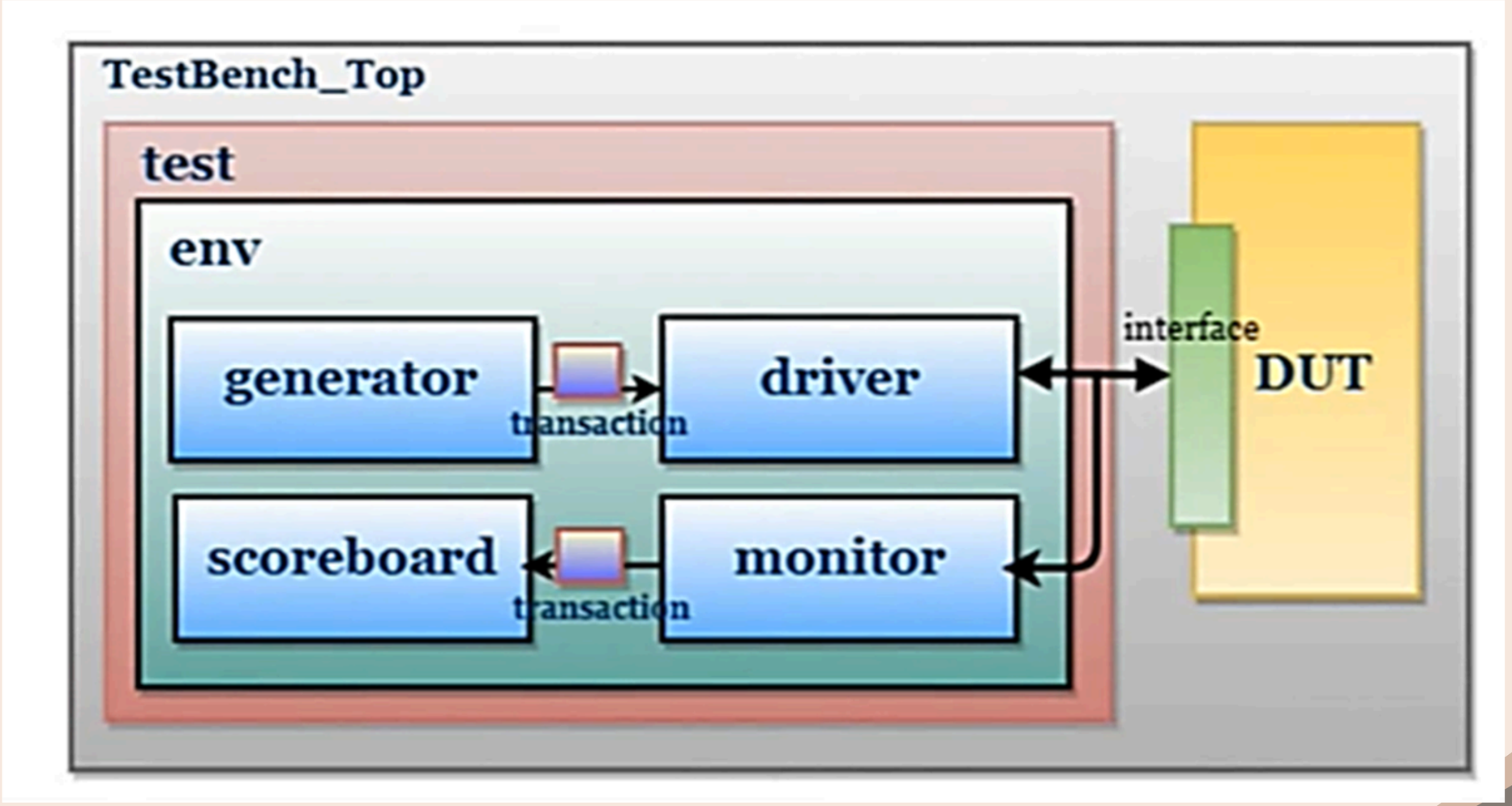
OBJECTIVE

We have selected serial in parallel out shift register for our project and have written a layered testbench for the same. The primary goal is to verify the functionality of a serial-in parallel-out shift register.
The testbench ensures that the design correctly shifts input serial data on the clock's rising edge and outputs the corresponding parallel data. By integrating key components like a generator, driver, monitor, and scoreboard, the testbench validates the design under diverse input scenarios, guaranteeing reliability and adherence to design specifications. This modular approach emphasizes the importance of structured testbench development in contemporary digital verification

BLOCK DIAGRAM OF SIPO



LAYERED TESTBENCH BLOCK DIAGRAM



Now let’s look at the DUT and layered testbench codes with respect to the SIPO(serial in parallel out).

Starting with the DUT code

```
module serial_in_parallel_out #(parameter WIDTH = 4)
 ( input logic clk,
   input logic reset,
   input logic ctr,  //for shift direction i.e right or left (0=left,1=right)
   input logic serial_in, // serial input
   output logic [WIDTH-1:0] par_out); //parallel output


    always_ff @(posedge clk or posedge reset)
      begin
        if (reset)
          begin
            par_out <= '0;
          end

        else
          begin
            if(ctr == 0)      //left shift
          begin
          par_out <= {par_out[WIDTH-2:0], serial_in};
          end
        else                      //right shift
          begin
          par_out <= {serial_in, par_out[WIDTH-1:1]};
          end
          end
    end
endmodule
```

Now the interface block

```
interface sipo_interface #(parameter WIDTH = 4)
  (input logic clk,
   input logic reset);

    logic ctr;
    logic serial_in;
    logic [WIDTH-1:0] par_out;

endinterface
```

Following the interface we'll get into the transaction/packet class

```
class sipo_transaction;
    rand bit ctr;              // all the input and output signals are declared
    rand bit [3:0] serial_in;
    bit clk;
    bit reset;
    bit [3:0] par_out;


//Function for Displaying values of ctr , serial_in and par_out
  function void display(string name);
    $display("-------------------------");
    $display(" %s ",name);
    $display("-------------------------");
    $display("ctr = %0d,   serial_in = %0d",ctr,serial_in);
    $display("par_out=%d",par_out);
    $display("-------------------------");
  endfunction

endclass
```

Generator class

```
class sipo_generator;

  sipo_transaction trans;
  mailbox trans_mailbox;    //creating a mailbox

  function new(mailbox  mbox);        // custom constructorc
        trans_mailbox = mbox;
    endfunction

  task generating();   //generating input signals using randomize

      repeat(1)
      begin
        trans = new();
        trans.randomize();
        trans.display("Generator");
        trans_mailbox.put(trans);
      end
    endtask
endclass
```

Driver class

```
class sipo_driver;
    virtual sipo_interface vif;
    mailbox trans_mailbox;

    // Constructor
    function new(virtual sipo_interface vif,
                 mailbox mbox);
        this.vif = vif;
        this.trans_mailbox = mbox;
    endfunction

    // Drive transactions onto the DUT
    task drive();

      repeat(1)
      begin
       sipo_transaction trans;

       trans_mailbox.get(trans);

        vif.ctr      <= trans.ctr;
        vif.serial_in <= trans.serial_in;
        vif.clk       <= trans.clk;

        trans.par_out = vif.par_out;
        trans.display("Driver");
        end
    endtask
endclass
```

Monitor Class

```
class sipo_monitor;
    virtual sipo_interface vif;
    mailbox montoscrboard;

    // Constructor
  function new(virtual sipo_interface vif,
               mailbox montoscrboard);
        this.vif = vif;
        this.montoscrboard = montoscrboard;
    endfunction

    // Monitor the DUT signals
    task monitor();

      repeat(1)
      #3;
      begin
          sipo_transaction trans;

      trans = new();
      trans.ctr       = vif.ctr;
      trans.serial_in       = vif.serial_in;
      trans.clk     = vif.clk;
      trans.par_out    = vif.par_out;
      trans.clk  = vif.clk;
      montoscrboard.put(trans);
      trans.display("Monitor");
    end

    endtask
endclass
```

Scoreboard class

```
class sipo_scoreboard #(parameter WIDTH=4);
    mailbox montoscrboard;
    virtual sipo_interface vif;
  logic [WIDTH-1:0] reference;

    // Constructor
    function new(mailbox mbox, virtual sipo_interface vif);
        this.montoscrboard = mbox;
        this.vif = vif;
        reference = '0;
    endfunction

    // Predict and compare results
    task check();

            sipo_transaction trans;
      repeat(1)
      begin
            montoscrboard.get(trans);
            // Updating reference based on ctr signal
          if (trans.ctr == 0)
            reference = {reference[WIDTH-2:0], trans.serial_in};
            else
              reference = {trans.serial_in, reference[WIDTH-1:1]};

            // Check DUT output
            @(posedge vif.clk);
        if (vif.par_out !== reference)
          $display("Mismatch: Expected %b, Got %b", reference, vif.par_out);
            else
              $display("Match: Expected %b, Got %b", reference, vif.par_out);
        trans.display("Scoreboard");
        end
    endtask
endclass
```

Now comes the environment class in which all the previous classes are present/instantiated

```
`include "sipo_transaction.sv"
`include "sipo_generator.sv"
`include "sipo_driver.sv"
`include "sipo_monitor.sv"
`include "sipo_scoreboard.sv"


class sipo_environment;
    sipo_generator generator;
    sipo_driver driver;
    sipo_monitor monitor;
    sipo_scoreboard scoreboard;
    mailbox driver_mbox, monitor_mbox;
    virtual sipo_interface vif;

    // Constructor
    function new(virtual sipo_interface vif);
        this.vif = vif;
        driver_mbox = new();
        monitor_mbox = new();
        generator = new(driver_mbox);
        driver = new(vif, driver_mbox);
        monitor = new(vif, monitor_mbox);
        scoreboard = new(monitor_mbox, vif);
    endfunction

    // Run environment tasks
    task test();
        fork
            generator.generating();
            driver.drive();
            monitor.monitor();
            scoreboard.check();
        join
    endtask
```

Test module

```
`include "sipo_environment.sv"

program test(sipo_interface i_intf);
  sipo_environment env;

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

endprogram
```

Finally the testbench module

```
`include "sipo_interface.sv"
`include "test.sv"

module tb_serial_in_parallel_out();


    // Interface
  sipo_interface sif();
  test t1(sif);

    // DUT instantiation
    serial_in_parallel_out dut
  (    .clk(sif.clk),
        .reset(sif.reset),
      .ctr(sif.ctr),
        .serial_in(sif.serial_in),
      .par_out(sif.par_out)
    );


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

endmodule
```

THANK YOU