```systemverilog
class transaction;

 typedef enum bit [1:0] {write = 2'b00 , read = 2'b01} oper_type;

  randc oper_type oper;

  bit rx;

  rand bit [7:0] dintx;

  bit send;
  bit tx;

  bit [7:0] doutrx;
  bit donetx;
  bit donerx;



  function void display (input string tag);
    $display("[%0s] : oper : %0s send : %0b TX_DATA : %b RX_IN : %0b TX_OUT : %0b RX_OUT
    : %b DONE_TX : %0b DONE_RX : %0b", tag, oper.name(), send, dintx, rx, tx, doutrx,
    donetx, donerx);
  endfunction

  function transaction copy();
    copy = new();
    copy.rx = this.rx;
    copy.dintx = this.dintx;
    copy.send = this.send;
    copy.tx = this.tx;
    copy.doutrx = this.doutrx;
    copy.donetx = this.donetx;
    copy.donerx = this.donerx;
    copy.oper = this.oper;
  endfunction

endclass


/*
module tb;
  transaction tr;


  initial begin
    tr = new();
    tr.display("TOP");

  end


endmodule

*/


class generator;

 transaction tr;

  mailbox #(transaction) mbx;

  event done;

  int count = 0;

  event drvnext;
  event sconext;


  function new(mailbox #(transaction) mbx);
    this.mbx = mbx;
    tr = new();
  endfunction


  task run();

    repeat(count) begin
      assert(tr.randomize) else $error("[GEN] :Randomization Failed");
      mbx.put(tr.copy);
      tr.display("GEN");
      @(drvnext);
      @(sconext);
    end

    -> done;
  endtask


endclass

////////////////////////////////////////////////////////

class driver;


  virtual uart_if vif;

  transaction tr;

  mailbox #(transaction) mbx;

  mailbox #(bit [7:0]) mbxds;


  event drvnext;

  bit [7:0] din;


  bit wr = 0;  ///random operation read / write
  bit [7:0] datarx;  ///data rcvd during read





  function new(mailbox #(bit [7:0]) mbxds, mailbox #(transaction) mbx);
    this.mbx = mbx;
    this.mbxds = mbxds;
   endfunction



  task reset();
    vif.rst <= 1'b1;
    vif.dintx <= 0;
    vif.send <= 0;
    vif.rx <= 1'b1;
    vif.doutrx <= 0;
    vif.tx <= 1'b1;
    vif.donerx <= 0;
    vif.donetx <= 0;
    repeat(5) @(posedge vif.uclktx);
    vif.rst <= 1'b0;
    @(posedge vif.uclktx);
    $display("[DRV] : RESET DONE");
  endtask



  task run();

    forever begin
      mbx.get(tr);

      if(tr.oper == 2'b00)  ////data transmission
          begin
          //
            @(posedge vif.uclktx);
            vif.rst <= 1'b0;
            vif.send <= 1'b1;  ///start data sending op
            vif.rx <= 1'b1;
            vif.dintx = tr.dintx;
            @(posedge vif.uclktx);
            vif.send <= 1'b0;
              ////wait for completion
            //repeat(9) @(posedge vif.uclktx);
            mbxds.put(tr.dintx);
            $display("[DRV]: Data Sent : %0d", tr.dintx);
             wait(vif.donetx == 1'b1);
             ->drvnext;
          end

      else if (tr.oper == 2'b01)
               begin

                 @(posedge vif.uclkrx);
                  vif.rst <= 1'b0;
                  vif.rx <= 1'b0;
                  vif.send <= 1'b0;
                  @(posedge vif.uclkrx);

                 for(int i=0; i<=7; i++)
                 begin
                      @(posedge vif.uclkrx);
                      vif.rx <= $urandom;
                      datarx[i] = vif.rx;
                 end


                mbxds.put(datarx);

                $display("[DRV]: Data RCVD : %0d", datarx);
                wait(vif.donerx == 1'b1);
                 vif.rx <= 1'b1;
                ->drvnext;


             end



    end

  endtask

endclass

/*

module tb;
  generator gen;
  driver drv;
  event next;
  event done;
  mailbox #(transaction) mbx;
  mailbox #(bit [7:0]) mbxt;

  uart_if vif();

  uart_top #(1000000, 9600) dut
  (vif.clk,vif.rst,vif.rx,vif.dintx,vif.send,vif.tx,vif.doutrx,vif.donetx, vif.donerx);



    initial begin
      vif.clk <= 0;
    end

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



  initial begin
    mbx = new();
    mbxt = new();
    gen = new(mbx);
    drv = new(mbxt,mbx);
    gen.count  = 20;
    drv.vif = vif;

    gen.drvnext = next;

    drv.drvnext = next;
  end

  initial begin

    fork
      gen.run();
      drv.run();
    join_none
    wait(gen.done.triggered);
    $finish();
  end

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



assign vif.uclktx = dut.utx.uclk;
assign vif.uclkrx = dut.rtx.uclk;



endmodule

*/

class monitor;

  transaction tr;

  mailbox #(bit [7:0]) mbx;

  bit [7:0] srx; //////send
  bit [7:0] rrx; ///// recv



  virtual uart_if vif;


  function new(mailbox #(bit [7:0]) mbx);
    this.mbx = mbx;
    endfunction

  task run();

    forever begin

       @(posedge vif.uclktx);
      if ( (vif.send== 1'b1) && (vif.rx == 1'b1) )
                  begin

                    @(posedge vif.uclktx); ////start collecting tx data from next clock
                    tick

                for(int i = 0; i<= 7; i++)
                begin
                      @(posedge vif.uclktx);
                      srx[i] = vif.tx;

                end


                    $display("[MON] : DATA SEND on UART TX %0d", srx);

                    //////////wait for done tx before proceeding next
                    transaction
                  @(posedge vif.uclktx); //
                  mbx.put(srx);

                 end

      else if ((vif.rx == 1'b0) && (vif.send == 1'b0) )
        begin
          wait(vif.donerx == 1);
           rrx = vif.doutrx;
           $display("[MON] : DATA RCVD RX %0d", rrx);
           @(posedge vif.uclktx);
           mbx.put(rrx);
      end
  end
endtask


endclass

////////////////////////////////////////////////////////

/*
module tb;
  generator gen;
  driver drv;
  monitor mon;

  event sconext;
  event drvnext;

  event done;

  mailbox #(transaction) mbx;
  mailbox #(bit [7:0]) mbxds;
  mailbox #(bit [7:0]) mbxms;

  uart_if vif();

  uart_top #(1000000, 9600) dut
```

```
    (vif.clk,vif.rst,vif.rx,vif.dintx,vif.send,vif.tx,vif.doutrx,vif.donetx, vif.donerx);



      initial begin
        vif.clk <= 0;
      end

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



    initial begin
      mbx = new();
      mbxds = new();
      mbxms = new();


      gen = new(mbx);
      drv = new(mbxds,mbx);
      mon= new(mbxms);

      gen.count  = 10;
      drv.vif = vif;
      mon.vif = vif;

      gen.drvnext = drvnext;

      drv.drvnext = drvnext;

      gen.sconext = sconext;

      mon.sconext = sconext;
    end

    initial begin

      fork
        gen.run();
        drv.run();
        mon.run();
      join_none
      wait(gen.done.triggered);
      $finish();
    end

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



  assign vif.uclktx = dut.utx.uclk;
  assign vif.uclkrx = dut.rtx.uclk;



  endmodule
  */

  //////////////////////////////////////////////////////////////////////////



  class scoreboard;
    mailbox #(bit [7:0]) mbxds, mbxms;

    bit [7:0] ds;
    bit [7:0] ms;

     event sconext;

    function new(mailbox #(bit [7:0]) mbxds, mailbox #(bit [7:0]) mbxms);
      this.mbxds = mbxds;
      this.mbxms = mbxms;
    endfunction

    task run();
      forever begin

        mbxds.get(ds);
        mbxms.get(ms);

        $display("[SCO] : DRV : %0d MON : %0d", ds, ms);
        if(ds == ms)
          $display("DATA MATCHED");
        else
          $display("DATA MISMATCHED");

       ->sconext;
      end
    endtask


  endclass

  ///////////////////////////////

  class environment;

      generator gen;
      driver drv;
      monitor mon;
      scoreboard sco;



      event nextgd; ///gen -> drv

      event nextgs;  /// gen -> sco

    mailbox #(transaction) mbxgd; ///gen - drv

    mailbox #(bit [7:0]) mbxds; /// drv - sco


    mailbox #(bit [7:0]) mbxms;  /// mon - sco

      virtual uart_if vif;


    function new(virtual uart_if vif);

      mbxgd = new();
      mbxms = new();
      mbxds = new();

      gen = new(mbxgd);
      drv = new(mbxds,mbxgd);



      mon = new(mbxms);
      sco = new(mbxds, mbxms);

      this.vif = vif;
      drv.vif = this.vif;
      mon.vif = this.vif;

      gen.sconext = nextgs;
      sco.sconext = nextgs;

      gen.drvnext = nextgd;
      drv.drvnext = nextgd;

    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);
      $finish();
    endtask

    task run();
      pre_test();
      test();
      post_test();
    endtask



  endclass

  ///////////////////////////////////////////


  module tb;

    uart_if vif();

    uart_top #(1000000, 9600) dut
    (vif.clk,vif.rst,vif.rx,vif.dintx,vif.send,vif.tx,vif.doutrx,vif.donetx, vif.donerx);



      initial begin
        vif.clk <= 0;
      end

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

      environment env;



      initial begin
        env = new(vif);
        env.gen.count = 5;
        env.run();
      end


      initial begin
        $dumpfile("dump.vcd");
```

```
        $dumpvars;
      end

    assign vif.uclktx = dut.utx.uclk;
    assign vif.uclkrx = dut.rtx.uclk;

    endmodule



  ////////////////////////////////////////


```