DAY-89
#100DAYSRTL

"Aim":-Verification of UART Protocol using System Verilog

"TestBench Codes":-

Design Code:-

```
module uart_top #(parameter clk_freq = 1000000,parameter baud_rate = 9600)
(
  input clk,rst,
  input rx,
  input [7:0] dintx,
  input newd,
  output tx,
  output [7:0] doutrx,
  output donetx,
  output donerx
    );
  uarttx #(clk_freq, baud_rate) utx (clk, rst, newd, dintx, tx, donetx);
  uartrx #(clk_freq, baud_rate) rtx(clk, rst, rx, donerx, doutrx);
endmodule

module uarttx #(parameter clk_freq = 1000000,parameter baud_rate = 9600)
(
input clk,rst,
input newd,
input [7:0] tx_data,
output reg tx,
output reg donetx
);
  localparam clkcount = (clk_freq/baud_rate); ///x
integer count = 0;
integer counts = 0;
reg uclk = 0;
enum bit[1:0] {idle = 2'b00, start = 2'b01, transfer = 2'b10, done = 2'b11} state;
 ////////////uart_clock_gen
  always@(posedge clk)
    begin
      if(count < clkcount/2)
        count <= count + 1;
      else begin
        count <= 0;
        uclk <= ~uclk;
      end
    end
  reg [7:0] din;
  ////////////////////Reset decoder
  always@(posedge uclk)
    begin
      if(rst)
      begin
        state <= idle;
      end
     else
     begin
     case(state)
       idle:
         begin
           counts <= 0;
           tx <= 1'b1;
           donetx <= 1'b0;

           if(newd)
           begin
             state <= transfer;
             din <= tx_data;
             tx <= 1'b0;
           end
           else
             state <= idle;
         end
      transfer: begin
        if(counts <= 7) begin
           counts <= counts + 1;
           tx <= din[counts];
           state <= transfer;
        end
        else
        begin
           counts <= 0;
           tx <= 1'b1;
           state <= idle;
          donetx <= 1'b1;
        end
      end
      default : state <= idle;
    endcase
  end
end
endmodule
```

```
module uartrx #(parameter clk_freq = 1000000,parameter baud_rate = 9600 )
 (
input clk,
input rst,
input rx,
output reg done,
output reg [7:0] rxdata
);
localparam clkcount = (clk_freq/baud_rate);
integer count = 0;
integer counts = 0;
reg uclk = 0;
enum bit[1:0] {idle = 2'b00, start = 2'b01} state;
 ////////////uart_clock_gen
  always@(posedge clk)
    begin
      if(count < clkcount/2)
        count <= count + 1;
      else begin
        count <= 0;
        uclk <= ~uclk;
      end
    end
  always@(posedge uclk)
    begin
      if(rst)
      begin
     rxdata <= 8'h00;
     counts <= 0;
     done <= 1'b0;
      end
     else
     begin
     case(state)
     idle :
     begin
     rxdata <= 8'h00;
     counts <= 0;
     done <= 1'b0;
     if(rx == 1'b0)
       state <= start;
     else
       state <= idle;
     end
     start:
     begin
       if(counts <= 7)
      begin
     counts <= counts + 1;
     rxdata <= {rx, rxdata[7:1]};
     end
     else
     begin
     counts <= 0;
     done <= 1'b1;
     state <= idle;
     end
     end
   default : state <= idle;
   endcase
end
end
endmodule
```

“Interface”:-

```
interface uart_if;
  logic clk;
  logic uclktx;
  logic uclkrx;
  logic rst;
  logic rx;
  logic [7:0] dintx;
  logic newd;
  logic tx;
  logic [7:0] doutrx;
  logic donetx;
  logic donerx;
endinterface
```

“Transaction”:-

```
class transaction;
  typedef enum bit  {write = 1'b0 , read = 1'b1} oper_type;
  randc oper_type oper;
  bit rx;
  rand bit [7:0] dintx;
  bit newd;
  bit tx;
  bit [7:0] doutrx;
  bit donetx;
  bit donerx;
  function transaction copy();
    copy = new();
    copy.rx = this.rx;
    copy.dintx = this.dintx;
    copy.newd = this.newd;
    copy.tx = this.tx;
    copy.doutrx = this.doutrx;
    copy.donetx = this.donetx;
    copy.donerx = this.donerx;
    copy.oper = this.oper;
  endfunction
endclass
```

“Generator”:-

```
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);
      $display("[GEN]: Oper : %0s Din : %0d",tr.oper.name(), tr.dintx);
      @(drvnext);
      @(sconext);
    end
    -> done;
  endtask
endclass
```

“Driver”:-

```
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.newd <= 0;
    vif.rx <= 1'b1;
    repeat(5) @(posedge vif.uclktx);
    vif.rst <= 1'b0;
    @(posedge vif.uclktx);
    $display("[DRV] : RESET DONE");
    $display("----------------------------------------");
  endtask
  task run();
    forever begin
      mbx.get(tr);
      if(tr.oper == 1'b0)  ////data transmission
          begin
          //
            @(posedge vif.uclktx);
            vif.rst <= 1'b0;
            vif.newd <= 1'b1;  ///start data sending op
            vif.rx <= 1'b1;
            vif.dintx = tr.dintx;
            @(posedge vif.uclktx);
            vif.newd <= 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 == 1'b1)
                begin
                  @(posedge vif.uclkrx);
                   vif.rst <= 1'b0;
                   vif.rx <= 1'b0;
                   vif.newd <= 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
```

“Monitor”:-

```
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.newd== 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.newd == 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
```

“Scoreboard”:-

```
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");
      $display("-------------------------------------");
     ->sconext;
    end
  endtask
endclass
```

“Environment”:-

```
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
```

“Tb_Top”:-

```
module tb;
  uart_if vif();
  uart_top #(1000000, 9600) dut (vif.clk,vif.rst,vif.rx,vif.dintx,vif.newd,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
```

“Result”:-

```
# KERNEL: [DRV] : RESET DONE
# KERNEL: ----------------------------------------
# KERNEL: [GEN]: Oper : read Din : 88
# KERNEL: [DRV]: Data RCVD : 98
# KERNEL: [MON] : DATA RCVD RX 98
# KERNEL: [SCO] : DRV : 98 MON : 98
# KERNEL: DATA MATCHED
# KERNEL: ----------------------------------------
# KERNEL: [GEN]: Oper : write Din : 10
# KERNEL: [DRV]: Data Sent : 10
# KERNEL: [MON] : DATA SEND on UART TX 10
# KERNEL: [SCO] : DRV : 10 MON : 10
# KERNEL: DATA MATCHED
# KERNEL: ----------------------------------------
# KERNEL: [GEN]: Oper : write Din : 39
# KERNEL: [DRV]: Data Sent : 39
# KERNEL: [MON] : DATA SEND on UART TX 39
# KERNEL: [SCO] : DRV : 39 MON : 39
# KERNEL: DATA MATCHED
# KERNEL: ----------------------------------------
# KERNEL: [GEN]: Oper : read Din : 254
# KERNEL: [DRV]: Data RCVD : 224
# KERNEL: [MON] : DATA RCVD RX 224
# KERNEL: [SCO] : DRV : 224 MON : 224
# KERNEL: DATA MATCHED
# KERNEL: ----------------------------------------
# KERNEL: [GEN]: Oper : write Din : 173
# KERNEL: [DRV]: Data Sent : 173
# KERNEL: [MON] : DATA SEND on UART TX 173
# KERNEL: [SCO] : DRV : 173 MON : 173
# KERNEL: DATA MATCHED
# KERNEL: ----------------------------------------
```