Momen Mostafa Mohamed Elzaghawy

UART_TX UVM

Design:

```
module UART (UART_if.DUT uartif);

    reg [3:0] state = uartif.IDLE;
    reg [3:0] counter = 0;
    reg [7:0] DATA_reg;
    reg  PARITY_bit;
always @(posedge uartif.clk or negedge uartif.reset) begin
    if (!uartif.reset) begin
        state <= uartif.IDLE;
        uartif.TX_OUT <= 1'b1;
        uartif.Busy <= 1'b0;
        counter <= 0;
    end else begin
        case (state)
          uartif.IDLE: begin
                uartif.TX_OUT <= 1'b1;
                uartif.Busy <= 1'b0;
                counter <= 0;
                if (uartif.DATA_VALID) begin
                    DATA_reg <= uartif.P_DATA;
                    PARITY_bit <= (uartif.PAR_TYP == 0) ? ~^uartif.P_DATA : ^uartif.P_DATA;
                    state <= uartif.START;
                    uartif.Busy <= 1'b1;
                end
          end
          uartif.START: begin
                uartif.TX_OUT <= 1'b0;
                state <= uartif.DATA;
                counter <= 0;
          end
          uartif.DATA: begin
                  uartif.TX_OUT <= DATA_reg[counter];
                  counter <= counter + 1;
                  if (counter == 7)
                  state <= (uartif.PAR_EN) ? uartif.PARITY : uartif.STOP;
          end
          uartif.PARITY: begin
                    uartif.TX_OUT <= PARITY_bit;
                    state <= uartif.STOP;
          end
          uartif.STOP: begin
                  uartif.TX_OUT <= 1'b1;
                  state <= uartif.IDLE;
          end
        endcase
        end
end
endmodule
```

Golden model:

```
module UART_golden_model(UART_if.GOLDEN uartif);
    reg [3:0] counter = 0;
    reg [7:0] DATA_reg;
    reg  PARITY_bit;
    reg [3:0]cs,ns;
   // cs state
   always @(posedge uartif.clk , negedge uartif.reset)begin
        if (!uartif.reset) begin
           cs <= uartif.IDLE;
           counter <= 0;
        end
       else
           cs <= ns;
   end
   //ns state
     always @(*)begin
           case(cs)
               uartif.IDLE : begin
                             if (uartif.DATA_VALID) begin
                             ns = uartif.START;
                           end
                         end
               uartif.START  : ns = uartif.DATA;
               uartif.DATA   :begin
                 if (counter == 7)
                 ns = (uartif.PAR_EN) ? uartif.PARITY : uartif.STOP;
         end
               uartif.PARITY : ns = uartif.STOP;
               uartif.STOP   : ns = uartif.IDLE;
               default : ns = uartif.IDLE;
           endcase
     end
```

```
33      //output
34      always @(posedge uartif.clk , negedge uartif.reset) begin
35        if (!uartif.reset) begin
36            uartif.TX_OUT_ex <= 1'b1;
37            uartif.Busy_ex <= 1'b0;
38            counter <= 0;
39         end
40         else begin
41         case (cs)
42          uartif.IDLE: begin
43                uartif.TX_OUT_ex <= 1'b1;
44                uartif.Busy_ex <= 1'b0;
45                counter <= 0;
46                if (uartif.DATA_VALID) begin
47                    DATA_reg <= uartif.P_DATA;
48                    PARITY_bit <= (uartif.PAR_TYP == 0) ? ~^uartif.P_DATA : ^uartif.P_DATA;
49                    uartif.Busy_ex <= 1'b1;
50                end
51          end
52          uartif.START: begin
53                uartif.TX_OUT_ex <= 1'b0;
54                counter <= 0;
55          end
56          uartif.DATA: begin
57                  uartif.TX_OUT_ex <= DATA_reg[counter];
58                  counter <= counter + 1;
59          end
60          uartif.PARITY: begin
61                    uartif.TX_OUT_ex <= PARITY_bit;
62          end
63          uartif.STOP: begin
64                  uartif.TX_OUT_ex <= 1'b1;
65          end
66         endcase
67          end
68      end
69  endmodule
```

SVA:

```
module UART_sva (UART_if.DUT uartif);

    always_comb begin
        if(!uartif.reset)
       assert_reset: assert final ((uartif.TX_OUT)&&(!uartif.Busy)&&(UART.state==uartif.IDLE)&&(UART.counter==0));
        cover_reset:cover final ((uartif.TX_OUT)&&(!uartif.Busy)&&(UART.state==uartif.IDLE)&&(UART.counter==0));
    end
    property no2;
    @(posedge uartif.clk) disable iff(!uartif.reset)
                    ((UART.state==uartif.IDLE) && (!uartif.DATA_VALID) )|=>
                     (uartif.TX_OUT == 1) && (!uartif.Busy) && (UART.counter == 1'b0) ;
    endproperty
      property no3;
      @(posedge uartif.clk) disable iff(!uartif.reset)
                    ((UART.state==uartif.IDLE) && (!uartif.DATA_VALID) )|=> (UART.state==uartif.IDLE) ;
     endproperty
     property no4;
    @(posedge uartif.clk) disable iff(!uartif.reset)
                    (UART.state==uartif.IDLE)&&(uartif.DATA_VALID) && (uartif.PAR_TYP==0) |=>
                     (UART.PARITY_bit==~^$past(uartif.P_DATA)) && (uartif.Busy) && (UART.DATA_reg==$past(uartif.P_DATA)) &&(UART.state==uartif.START);
    endproperty
     property no5;
    @(posedge uartif.clk) disable iff(!uartif.reset)
                    (UART.state==uartif.IDLE)&&(uartif.DATA_VALID) && (uartif.PAR_TYP==1) |=>
                     (UART.PARITY_bit==^$past(uartif.P_DATA)) && (uartif.Busy) && (UART.DATA_reg==$past(uartif.P_DATA)) &&(UART.state==uartif.START);
    endproperty

    property no6;
      @(posedge uartif.clk) disable iff(!uartif.reset)
                    (UART.state==uartif.IDLE)&&(uartif.DATA_VALID) |=> (UART.state==uartif.START);
      endproperty
    property no7;
    @(posedge uartif.clk) disable iff(!uartif.reset)
                    (UART.state==uartif.START) |=>
                     (uartif.TX_OUT==0) && (UART.state==uartif.DATA) && (UART.counter==0);
    endproperty
    property no8;
    @(posedge uartif.clk) disable iff(!uartif.reset)
                    ((UART.state==uartif.DATA) && (UART.counter != 7 )) |=>
                     ((uartif.TX_OUT)==UART.DATA_reg[$past(UART.counter)]) && (UART.counter==$past(UART.counter)+1);
    endproperty
    property no9;
      @(posedge uartif.clk) disable iff(!uartif.reset)
                    ((UART.state==uartif.DATA) && (UART.counter != 7 )) |=> (UART.state==uartif.DATA) ;

      endproperty
```

```
      property no10;
      @(posedge uartif.clk) disable iff(!uartif.reset)
                          (UART.state==uartif.DATA)&&(UART.counter==7)&&(uartif.PAR_EN==1) |=>
                            (UART.state==uartif.PARITY);
      endproperty
      property no11;
      @(posedge uartif.clk) disable iff(!uartif.reset)
                          (UART.state==uartif.DATA)&&(UART.counter==7)&&(uartif.PAR_EN==0) |=>
                            (UART.state==uartif.STOP);
      endproperty
      property no12;
      @(posedge uartif.clk) disable iff(!uartif.reset)
                          (UART.state==uartif.PARITY) |=>
                           (uartif.TX_OUT==$past(UART.PARITY_bit)) &&(UART.state==uartif.STOP);
      endproperty
       property no13;
      @(posedge uartif.clk) disable iff(!uartif.reset)
                          (UART.state==uartif.STOP) |=>
                           (uartif.TX_OUT==1) &&(UART.state==uartif.IDLE);
      endproperty

      assert_2:assert property (no2);
      cover_2:cover property (no2);
      assert_3:assert property (no3);
      cover_3:cover property (no3);
      assert_4:assert property (no4);
      cover_4:cover property (no4);
      assert_5:assert property (no5);
      cover_5:cover property (no5);
      assert_6:assert property (no6);
      cover_6:cover property (no6);
      assert_7:assert property (no7);
      cover_7:cover property (no7);
      assert_8:assert property (no8);
      cover_8:cover property (no8);
      assert_9:assert property (no9);
      cover_9:cover property (no9);
      assert_10:assert property (no10);
      cover_10:cover property (no10);
      assert_11:assert property (no11);
      cover_11:cover property (no11);
      assert_12:assert property (no12);
      cover_12:cover property (no12);
      assert_13:assert property (no13);
      cover_13:cover property (no13);


endmodule
```

Interface:

UART_if.sv

```
interface UART_if (clk);
    input bit clk;
    logic reset;
    logic [7:0] P_DATA;
    logic PAR_EN;
    logic PAR_TYP;
    logic DATA_VALID;
    logic TX_OUT , TX_OUT_ex;
    logic Busy , Busy_ex;

    parameter IDLE = 0, START = 1, DATA = 2, PARITY = 3, STOP = 4;

    modport DUT ( input clk , reset , P_DATA , PAR_EN , PAR_TYP , DATA_VALID ,
                  output TX_OUT , Busy);

    modport GOLDEN ( input clk , reset , P_DATA , PAR_EN , PAR_TYP , DATA_VALID ,
                    output TX_OUT_ex , Busy_ex);

    modport TEST ( output  reset , P_DATA , PAR_EN , PAR_TYP , DATA_VALID ,
                    input clk ,TX_OUT , Busy , TX_OUT_ex , Busy_ex);

    modport MONITOR ( output clk , reset , P_DATA , PAR_EN , PAR_TYP , DATA_VALID , TX_OUT , Busy ,TX_OUT_ex , Busy_ex);
endinterface


```

Top:

UART_top.sv

```
import uvm_pkg::*;
`include "uvm_macros.svh"
import UART_test_pkg::*;

module UART_top();

bit clk;

initial begin
    forever begin
        #1 clk = ~clk;
    end
end

UART_if uartif (clk);
UART dut (uartif);
UART_golden_model GM (uartif);
bind UART UART_sva UART_sva_inst(uartif);

initial begin
    uvm_config_db#(virtual UART_if)::set(null, "uvm_test_top", "UART_IF", uartif);
    run_test("UART_test");
end

endmodule
```

Test:

```
package UART_test_pkg;
import UART_env_pkg::*;
import UART_config_pkg::*;
import UART_seq_pkg::*;
import uvm_pkg::*;
import UART_seq_item_pkg::*;
`include "uvm_macros.svh"

class UART_test extends uvm_test;
    `uvm_component_utils(UART_test)

    UART_env env;
    UART_config UART_cfg;
    virtual UART_if UART_vif;
    UART_main_seq main_seq;
    UART_reset_seq reset_seq;

    function new(string name = "UART_test", uvm_component parent = null);
        super.new(name, parent);
    endfunction

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        env = UART_env::type_id::create("env", this);
        UART_cfg = UART_config::type_id::create("UART_cfg");
        main_seq = UART_main_seq::type_id::create("main_seq");
        reset_seq = UART_reset_seq::type_id::create("reset_seq");

        UART_cfg.is_active = UVM_ACTIVE;

        if (!uvm_config_db#(virtual UART_if)::get(this, "", "UART_IF", UART_cfg.UART_vif))
            `uvm_fatal("build_phase", "test - unable to get the virtual interface");

        uvm_config_db#(UART_config)::set(this, "*", "CFG", UART_cfg);
    endfunction

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        phase.raise_objection(this);


        `uvm_info("run_phase", "reset asserted", UVM_LOW)
        reset_seq.start(env.agt.sqr);
        `uvm_info("run_phase", "reset deasserted", UVM_LOW)


        `uvm_info("run_phase", "stimulus generation started 1", UVM_LOW)
        main_seq.start(env.agt.sqr);
        `uvm_info("run_phase", "stimulus generation ended 1", UVM_LOW)


        `uvm_info("run_phase", "reset asserted", UVM_LOW)
        reset_seq.start(env.agt.sqr);
        `uvm_info("run_phase", "reset deasserted", UVM_LOW)

        phase.drop_objection(this);
    endtask

endclass
endpackage
```

Agent:

```
package UART_agent_pkg;
import UART_driver_pkg::*;
import UART_monitor_pkg::*;
import UART_sequencer_pkg::*;
import UART_seq_item_pkg::*;
import UART_config_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"

class UART_agent extends uvm_agent;
    `uvm_component_utils(UART_agent)

    UART_sequencer sqr;
    UART_driver drv;
    UART_monitor mon;
    UART_config UART_cfg;
    uvm_analysis_port #(UART_seq_item) agt_ap;

    function new (string name = "UART_agent", uvm_component parent = null);
        super.new(name, parent);
    endfunction

    function void build_phase (uvm_phase phase);
        super.build_phase(phase);
        if (!uvm_config_db #(UART_config)::get(this, "", "CFG", UART_cfg))
            `uvm_fatal("build_phase", "test - unable to get the configuration");

        if (UART_cfg.is_active == UVM_ACTIVE) begin
            sqr = UART_sequencer::type_id::create("sqr", this);
            drv = UART_driver::type_id::create("drv", this);
        end
        mon = UART_monitor::type_id::create("mon", this);
        agt_ap = new("agt_ap", this);
    endfunction
        function void connect_phase(uvm_phase phase);
        mon.UART_vif = UART_cfg.UART_vif;
        mon.mon_ap.connect(agt_ap);
        if (UART_cfg.is_active == UVM_ACTIVE) begin
            drv.UART_vif = UART_cfg.UART_vif;
            drv.seq_item_port.connect(sqr.seq_item_export);
        end
    endfunction
endclass
endpackage
```

Env:

```
package UART_env_pkg;
import UART_agent_pkg::*;
import UART_scoreboard_pkg::*;
import UART_coverage_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"

class UART_env extends uvm_env;
    `uvm_component_utils(UART_env)

    UART_agent agt;
    UART_scoreboard sb;
    UART_coverage cov;

    function new(string name = "UART_env", uvm_component parent = null);
        super.new(name, parent);
    endfunction

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        agt = UART_agent::type_id::create("agt", this);
        sb = UART_scoreboard::type_id::create("sb", this);
        cov = UART_coverage::type_id::create("cov", this);
    endfunction

    function void connect_phase(uvm_phase phase);
        agt.agt_ap.connect(sb.sb_export);
        agt.agt_ap.connect(cov.cov_export);
    endfunction
endclass
endpackage
```

Config:

UART_config.sv

```
package UART_config_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"

class UART_config extends uvm_object;
    `uvm_object_utils(UART_config)

    virtual UART_if UART_vif;
    uvm_active_passive_enum is_active;

    function new(string name = "UART_config");
        super.new(name);
    endfunction
endclass
endpackage
```

Sequence item:

```
package UART_seq_item_pkg;
import uvm_pkg::*;
`include "uvm_macros.svh"

parameter IDLE = 0, START = 1, DATA = 2, PARITY = 3, STOP = 4;

class UART_seq_item extends uvm_sequence_item;
    `uvm_object_utils(UART_seq_item)

     bit clk;
    rand logic reset;
    rand logic [7:0] P_DATA;
    rand logic PAR_EN;
    rand logic PAR_TYP;
    rand logic DATA_VALID;
    rand int pattern_type;

     logic TX_OUT , TX_OUT_ex;
     logic Busy , Busy_ex;

     function new(string name = "UART_seq_item");
         super.new(name);
     endfunction

     function string convert2string();
         return $sformatf("%s reset=%b P_DATA=%h PAR_EN=%b PAR_TYP=%b DATA_VALID=%b",
                          super.convert2string(), reset, P_DATA, PAR_EN, PAR_TYP, DATA_VALID);
     endfunction

     function string convert2string_stimulus();
         return $sformatf("reset=%b P_DATA=%h PAR_EN=%b PAR_TYP=%b DATA_VALID=%b",
                          reset, P_DATA, PAR_EN, PAR_TYP, DATA_VALID);
     endfunction

   constraint reset_con {reset dist {0 := 3 , 1 := 97};}
   constraint PAR_TYP_con {PAR_TYP dist {0 := 50 , 1 := 50};}
   constraint PAR_EN_con {PAR_EN dist{0 := 75 , 1 := 25};}
   constraint DATA_VALID_con {DATA_VALID dist {0 := 8 , 1 := 92};} ;
   constraint P_DATA_LSB_con {P_DATA[0] dist {0 := 20 , 1:=80};}
   constraint pattern_type_con { pattern_type dist {0 := 96 , 1 := 4 };}
   constraint P_DATA_con { if (pattern_type) P_DATA inside{8'hFF , 8'h00 , 8'hAA};}

endclass
endpackage
```

Sequence:

```systemverilog
package UART_seq_pkg;
import uvm_pkg::*;
import UART_seq_item_pkg::*;
`include "uvm_macros.svh"

class UART_reset_seq extends uvm_sequence #(UART_seq_item);
    `uvm_object_utils(UART_reset_seq)
    UART_seq_item seq_item;

    function new(string name = "UART_reset_seq");
        super.new(name);
    endfunction

    task body;
        seq_item = UART_seq_item::type_id::create("seq_item");
        start_item(seq_item);
        seq_item.reset = 0;
        seq_item.P_DATA = 0;
        seq_item.PAR_EN = 0;
        seq_item.PAR_TYP = 0;
        seq_item.DATA_VALID = 0;
        finish_item(seq_item);
    endtask
endclass

class UART_main_seq extends uvm_sequence #(UART_seq_item);
    `uvm_object_utils(UART_main_seq)
    UART_seq_item seq_item;

    function new(string name = "UART_main_seq");
        super.new(name);
    endfunction

    task body;
        seq_item = UART_seq_item::type_id::create("seq_item");
        repeat (1000) begin
            start_item(seq_item);
            assert(seq_item.randomize());
            finish_item(seq_item);
        end
    endtask
endclass
endpackage
```

Sequencer:

```systemverilog
package UART_sequencer_pkg;
import uvm_pkg::*;
import UART_seq_item_pkg::*;
`include "uvm_macros.svh"

class UART_sequencer extends uvm_sequencer #(UART_seq_item);
    `uvm_component_utils(UART_sequencer);

    function new (string name = "UART_sequencer", uvm_component parent = null);
        super.new(name, parent);
    endfunction
endclass

endpackage
```

Driver:

```
package UART_driver_pkg;
import uvm_pkg::*;
import UART_seq_item_pkg::*;
`include "uvm_macros.svh"

class UART_driver extends uvm_driver #(UART_seq_item);
    `uvm_component_utils(UART_driver)

    virtual UART_if UART_vif;
    UART_seq_item stim_seq_item;

    function new(string name = "UART_driver", uvm_component parent = null);
        super.new(name, parent);
    endfunction

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever begin
            stim_seq_item = UART_seq_item::type_id::create("stim_seq_item");
            seq_item_port.get_next_item(stim_seq_item);

            UART_vif.reset = stim_seq_item.reset;
            UART_vif.P_DATA = stim_seq_item.P_DATA;
            UART_vif.PAR_EN = stim_seq_item.PAR_EN;
            UART_vif.PAR_TYP = stim_seq_item.PAR_TYP;
            UART_vif.DATA_VALID = stim_seq_item.DATA_VALID;

            @(negedge UART_vif.clk);
            seq_item_port.item_done();
            `uvm_info("run_phase", stim_seq_item.convert2string_stimulus(), UVM_HIGH)
        end
    endtask
endclass
endpackage
```

Coverage:

```
package UART_coverage_pkg;
import UART_seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"

class UART_coverage extends uvm_component;
    `uvm_component_utils(UART_coverage)

    uvm_analysis_export #(UART_seq_item) cov_export;
    uvm_tlm_analysis_fifo #(UART_seq_item) cov_fifo;
    UART_seq_item seq_item_cov;

      covergroup cvr_gp ;
  reset_cp : coverpoint seq_item_cov.reset {
    bins zero = {0} ;
    bins one = {1} ;}
  PAR_TYP_cp : coverpoint seq_item_cov.PAR_TYP {
    bins zero = {0} ;
    bins one = {1} ;}
  PAR_EN_cp : coverpoint seq_item_cov.PAR_EN {
    bins zero = {0} ;
    bins one = {1} ;}
  P_DATA_cp : coverpoint seq_item_cov.P_DATA {
    bins all_values = {[0:255]};}
  DATA_VALID_cp : coverpoint seq_item_cov.DATA_VALID {
    bins zero = {0} ;
    bins one = {1} ;}
  TX_OUT_cp : coverpoint seq_item_cov.TX_OUT {
    bins zero = {0} ;
    bins one = {1} ;}
  Busy_cp : coverpoint seq_item_cov.Busy {
    bins zero = {0} ;
    bins one = {1} ;}
  PAR_EN_TYPE_cross : cross PAR_EN_cp , PAR_TYP_cp ;
  P_DATA_DATA_VALID_cross : cross P_DATA_cp , DATA_VALID_cp ;
  TX_OUT_Busy_cross : cross TX_OUT_cp , Busy_cp ;

  endgroup

    function new(string name = "UART_coverage", uvm_component parent = null);
        super.new(name, parent);
        cvr_gp = new();
    endfunction

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        cov_export = new("cov_export", this);
        cov_fifo = new("cov_fifo", this);
    endfunction

    function void connect_phase(uvm_phase phase);
        super.connect_phase(phase);
        cov_export.connect(cov_fifo.analysis_export);
    endfunction

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever begin
            cov_fifo.get(seq_item_cov);
            cvr_gp.sample();
        end
    endtask
endclass
endpackage
```

Scoreboard:

```
package UART_scoreboard_pkg;
import UART_seq_item_pkg::*;
import uvm_pkg::*;
`include "uvm_macros.svh"

class UART_scoreboard extends uvm_scoreboard;
    `uvm_component_utils(UART_scoreboard)

    uvm_analysis_export #(UART_seq_item) sb_export;
    uvm_tlm_analysis_fifo #(UART_seq_item) sb_fifo;
    UART_seq_item seq_item_sb;
    int error_count = 0;
    int correct_count = 0;

    function new(string name = "UART_scoreboard", uvm_component parent = null);
        super.new(name, parent);
    endfunction

    function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        sb_export = new("sb_export", this);
        sb_fifo = new("sb_fifo", this);
    endfunction

    function void connect_phase(uvm_phase phase);
        super.connect_phase(phase);
        sb_export.connect(sb_fifo.analysis_export);
    endfunction

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever begin
            sb_fifo.get(seq_item_sb);
            if ((seq_item_sb.TX_OUT != seq_item_sb.TX_OUT_ex) && (seq_item_sb.Busy != seq_item_sb.Busy_ex)) begin
                `uvm_error("run_phase", $sformatf("comparison failed while ref = %0d", seq_item_sb.TX_OUT_ex))
                `uvm_error("run_phase", $sformatf("comparison failed while ref = %0d", seq_item_sb.Busy_ex))

                error_count++;
            end
            else begin
                correct_count++;
            end
        end
    endtask

    function void report_phase(uvm_phase phase);
        super.report_phase(phase);
        `uvm_info("report_phase", $sformatf("correct = %0d", correct_count), UVM_MEDIUM)
        `uvm_info("report_phase", $sformatf("error = %0d", error_count), UVM_MEDIUM)
    endfunction
endclass
endpackage
```

Monitor:

UART_monitor.sv

```
package UART_monitor_pkg;
import uvm_pkg::*;
import UART_seq_item_pkg::*;
`include "uvm_macros.svh"

class UART_monitor extends uvm_monitor;
    `uvm_component_utils(UART_monitor)

    virtual UART_if UART_vif;
    UART_seq_item rsp_seq_item;
    uvm_analysis_port #(UART_seq_item) mon_ap;

    function new (string name = "UART_monitor", uvm_component parent = null);
        super.new(name, parent);
    endfunction

    function void build_phase (uvm_phase phase);
        super.build_phase(phase);
        mon_ap = new("mon_ap", this);
    endfunction

    task run_phase (uvm_phase phase);
        super.run_phase(phase);
        forever begin
            rsp_seq_item = UART_seq_item::type_id::create("rsp_seq_item");
            @(posedge UART_vif.clk);

            rsp_seq_item.reset = UART_vif.reset;
            rsp_seq_item.P_DATA = UART_vif.P_DATA;
            rsp_seq_item.PAR_EN = UART_vif.PAR_EN;
            rsp_seq_item.PAR_TYP = UART_vif.PAR_TYP;
            rsp_seq_item.DATA_VALID = UART_vif.DATA_VALID;

            mon_ap.write(rsp_seq_item);
            `uvm_info("run_phase", rsp_seq_item.convert2string_stimulus(), UVM_HIGH)
        end
    endtask
endclass

endpackage
```

Do file:

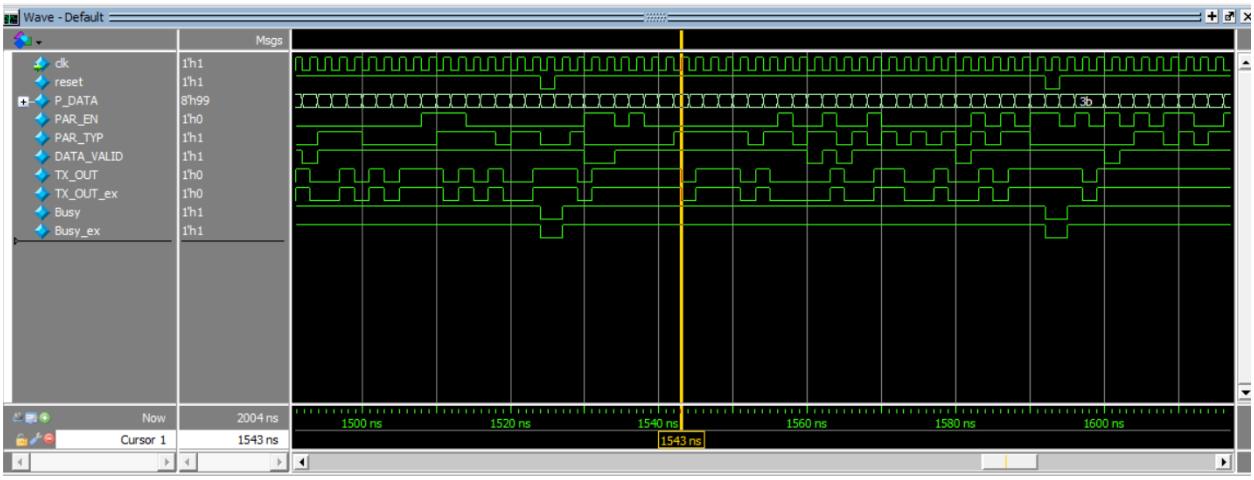
```
vlib work
vlog -f src_files_UART.list
vsim -voptargs=+acc work.UART_top -classdebug -uvmcontrol=all
add wave /UART_top/uartif/*
run -all
```

Src_files.list:

src_files_UART.list

```
UART_if.sv
UART.sv
UART_golden_model.sv
UART_config.sv
UART_seq_item.sv
UART_seq.sv
UART_sequencer.sv
UART_monitor.sv
UART_driver.sv
UART_scoreboard.sv
UART_coverage.sv
UART_agent.sv
UART_env.sv
UART_test.sv
UART_top.sv
```

Waveform and transcript:




```
UVM_INFO UART_test.sv(44) @ 2: uvm_test_top [run_phase] reset deasserted
UVM_INFO UART_test.sv(47) @ 2: uvm_test_top [run_phase] stimulus generation started 1
UVM_INFO UART_test.sv(49) @ 2002: uvm_test_top [run_phase] stimulus generation ended 1
UVM_INFO UART_test.sv(52) @ 2002: uvm_test_top [run_phase] reset asserted
UVM_INFO UART_test.sv(54) @ 2004: uvm_test_top [run_phase] reset deasserted
UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 2004: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
UVM_INFO UART_scoreboard.sv(48) @ 2004: uvm_test_top.env.sb [report_phase] correct = 1002
UVM_INFO UART_scoreboard.sv(49) @ 2004: uvm_test_top.env.sb [report_phase] error = 0

--- UVM Report Summary ---

** Report counts by severity
UVM_INFO :   12
UVM_WARNING :    0
UVM_ERROR :    0
UVM_FATAL :    0
** Report counts by id
[Questa UVM]     2
[RNTST]     1
[TEST_DONE]     1
[report_phase]     2
[run_phase]     6
** Note: $finish    : C:/questasim64_2024.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
   Time: 2004 ns  Iteration: 61  Instance: /UART_top
Break in Task uvm_pkg/uvm_root::run_test at C:/questasim64_2024.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh line 430
```