Abdalrahman Taha Mahmed

UART Using UVM

1.design

uart.sv

```
module uart( uart_if.DUT uartif);
parameter IDLE = 0, START = 1, DATA = 2, PARITY = 3, STOP = 4;
reg [3:0] state = IDLE;
reg [3:0] counter = 0;
reg [7:0] data_reg;
reg parity_bit;

always @(posedge uartif.clk or negedge uartif.rst_n) begin
  if(!uartif.rst_n) begin
      state <= IDLE;
      uartif.TX_OUT <= 1'b1;
      uartif.busy <= 1'b0;
      counter <= 0;
end
else begin
  case(state)
  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 <= START;
      uartif.busy <= 1'b1;
    end
  end
  START: begin
    uartif.TX_OUT <= 1'b0;
    state <= DATA;
    counter <= 0;
  end
  DATA: begin
    uartif.TX_OUT <= data_reg[counter];
    counter <= counter + 1;
    if (counter == 7)
        state <= (uartif.PAR_EN) ? PARITY : STOP;
  end
  PARITY: begin
    uartif.TX_OUT <= parity_bit;
    state <= STOP;
  end
  STOP: begin
    uartif.TX_OUT <= 1'b1;
    state <= IDLE;
  end
  endcase
end
```

2.golden model

uart_golden_model.sv

```
module uart_golden_model ( uart_if.GOLDEN_MODEL uartif);

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

reg [3:0] state = IDLE;
reg [3:0] counter = 0;
reg [7:0] data_reg;
reg parity_bit;

always @(posedge uartif.clk or negedge uartif.rst_n) begin
  if(!uartif.rst_n) begin
      state <= IDLE;
      uartif.TX_OUT_ex <= 1'b1;
      uartif.busy_ex <= 1'b0;
      counter <= 4'b0000;
end
else begin
  case(state)
  IDLE: begin
    uartif.TX_OUT_ex <= 1'b1;
    uartif.busy_ex <= 1'b0;
    counter <= 4'b0000;
    if (uartif.DATA_valid) begin
      data_reg <= uartif.P_DATA;
      parity_bit <= (uartif.PAR_TYP) ? ^uartif.P_DATA : ~^uartif.P_DATA;
      state <= START;
      uartif.busy_ex <= 1'b1;
    end
  end
  START: begin
    uartif.TX_OUT_ex <= 1'b0;
    state <= DATA;
    counter <= 4'b0000;
  end
  DATA: begin
    uartif.TX_OUT_ex <= data_reg[counter];
    counter <= counter + 1;
    if (counter == 7)
        state <= (uartif.PAR_EN) ? PARITY : STOP;
  end
```

```
  PARITY: begin
    uartif.TX_OUT_ex <= parity_bit;
    state <= STOP;
  end
  STOP: begin
    uartif.TX_OUT_ex <= 1'b1;
    state <= IDLE;
  end
  endcase
end
end
endmodule
```

3.sva

uart_sva.sv

```
module uart_sva ( uart_if.DUT uartif );

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

    property reset_check;
        @(posedge uartif.clk) (!uartif.rst_n) |=>
         (uart.state==IDLE && uartif.TX_OUT==1'b1 && uartif.busy==1'b0 && uart.counter==8'b00000000);
    endproperty
    assert property (reset_check) else $error(" error in rst ");
    cover property (reset_check);

    property check1;
        @(posedge uartif.clk) disable iff (!uartif.rst_n)
         (uart.state==IDLE)
         |=> (uartif.TX_OUT==1 && uartif.busy==0 && uart.counter==0);
    endproperty
    assert property (check2) else $error("error in check2");
    cover property (check2);

    property check2;
        @(posedge uartif.clk) disable iff (!uartif.rst_n)
         (uart.state==IDLE && uartif.DATA_valid)
         |=> (uart.data_reg==$past(uartif.P_DATA));
    endproperty
    assert property (check2) else $error("error in check2");
    cover property (check2);

    property check3;
        @(posedge uartif.clk) disable iff (!uartif.rst_n)
         (uart.state==IDLE && uartif.DATA_valid)
         |=> (uart.state==START && uartif.busy==1'b1);
    endproperty
    assert property (check3) else $error("error in check3");
    cover property (check3);

    property check4;
        @(posedge uartif.clk) disable iff (!uartif.rst_n)
         (uart.state==IDLE && uartif.DATA_valid && uartif.PAR_TYP)
         |=> (uart.parity_bit == ^$past(uartif.P_DATA));
    endproperty
    assert property (check4) else $error("error in check4");
    cover property (check4);
```

```
    property check5;
        @(posedge uartif.clk) disable iff (!uartif.rst_n)
         (uart.state==IDLE && uartif.DATA_valid && !uartif.PAR_TYP)
         |=> (uart.parity_bit == ~^$past(uartif.P_DATA));
    endproperty
    assert property (check5) else $error("error in check5");
    cover property (check5);

    property check6;
        @(posedge uartif.clk) disable iff (!uartif.rst_n)
         (uart.state==START)
         |=> (uartif.TX_OUT==1'b0 && uart.state==DATA && uart.counter==8'b00000000);
    endproperty
    assert property (check6) else $error("error in check6");
    cover property (check6);

    property check7;
        @(posedge uartif.clk) disable iff (!uartif.rst_n)
         (uart.state==DATA)
         |=> (uartif.TX_OUT==$past(uart.data_reg[uart.counter]));
    endproperty
    assert property (check7) else $error("error in check7");
    cover property (check7);

    property check8;
        @(posedge uartif.clk) disable iff (!uartif.rst_n)
         (uart.state==DATA)
         |=> (uart.counter==$past(uart.counter) + 1'b1);
    endproperty
    assert property (check8) else $error("error in check8");
    cover property (check8);

    property check9;
        @(posedge uartif.clk) disable iff (!uartif.rst_n)
         (uart.state==DATA && uart.counter == 7)
         |=> (uart.state== (uartif.PAR_EN) ? PARITY : STOP);
    endproperty
    assert property (check9) else $error("error in check9");
    cover property (check9);
```

```
    property check10;
        @(posedge uartif.clk) disable iff (!uartif.rst_n)
         (uart.state==PARITY)
         |=> (uartif.TX_OUT== $past(uart.parity_bit));
    endproperty
    assert property (check10) else $error("error in check10");
    cover property (check10);

    property check11;
        @(posedge uartif.clk) disable iff (!uartif.rst_n)
         (uart.state==PARITY)
         |=> (uart.state==STOP);
    endproperty
    assert property (check11) else $error("error in check11");
    cover property (check11);

    property check12;
        @(posedge uartif.clk) disable iff (!uartif.rst_n)
         (uart.state==STOP)
         |=> (uartif.TX_OUT==1'b1 && uart.state==IDLE);
    endproperty
    assert property (check12) else $error("error in check12");
    cover property (check12);

    property check13;
        @(posedge uartif.clk) disable iff (!uartif.rst_n)
         (uart.state==IDLE || uart.state==STOP)
         |=> (uartif.TX_OUT);
    endproperty
    assert property (check13) else $error("error in check13");
    cover property (check13);

endmodule
```

4.if

uart_if.sv

```
interface uart_if (clk);
    input bit clk;
    logic rst_n, PAR_EN, PAR_TYP, DATA_valid, TX_OUT, TX_OUT_ex, busy, busy_ex;
    logic [7:0] P_DATA;

    modport DUT (
    input clk, rst_n, PAR_EN, PAR_TYP, DATA_valid, P_DATA,
    output TX_OUT, busy
    );

    modport GOLDEN_MODEL (
    input clk, rst_n, PAR_EN, PAR_TYP, DATA_valid, P_DATA,
    output TX_OUT_ex, busy_ex
    );


endinterface
```

5.seq_items

uart_seq_item.sv

```
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)
    rand logic rst_n;
    rand logic [7:0] P_DATA;
    rand logic PAR_EN;
    rand logic PAR_TYP;
    rand logic DATA_valid;
    rand logic direction,cin;
    logic TX_OUT;
    logic TX_OUT_ex;
    logic busy;
    logic busy_ex;
    bit clk;
    bit [7:0] vals [] = '{8'b11111111 , 8'b00000000 , 8'b10101010};
    bit [7:0] P_DATA_vals;

  function new (string name = "uart_seq_item");
    super.new (name);
  endfunction
  function string convert2string();
      return $sformatf("%s rst_n=%b P_DATA=%b TX_OUT=%b busy=%b",super.convert2string(),rst_n,P_DATA,TX_OUT,busy);
  endfunction
  function string convert2string_stimulus();
      return $sformatf(" rst_n=%b P_DATA=%b TX_OUT=%b busy=%b",rst_n,P_DATA,TX_OUT,busy);
  endfunction
        constraint rst_cns{
            rst_n dist{0:= 3 ,1:= 97};
        }
        constraint PAR_TYP_cns{
            PAR_TYP dist{0:= 50 ,1:= 50};
        }
        constraint PAR_EN_cns{
            PAR_EN dist{0:= 75 ,1:= 25};
        }
        constraint DATA_valid_cns{
            DATA_valid dist{0:= 5 ,1:= 95};
        }
        constraint P_DATA_cns {
             P_DATA_vals inside {vals};
             P_DATA dist {
                8'b00001111 := 50,
                P_DATA_vals := 4,
                [1:255] := 50};
        }
```

6.scoreboard

uart_scoreboard.sv

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

class uart_sco extends uvm_scoreboard;
  `uvm_component_utils(uart_sco)

  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_sco" , 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("compartion failled while ref = %b",seq_item_sb.TX_OUT_ex))
      error_count++;
  end
  else
     correct_count++;
 end
endtask

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

7.config

uart_config.sv

```
package uart_conf_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
```

8.seq

uart_seq.sv

```
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.rst_n = 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 (9999) begin
       start_item(seq_item);
       assert (seq_item.randomize());
       finish_item (seq_item);
       end
    endtask
endclass

endpackage
```

9.sequencer

uart_sequencer.sv

```
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
```

10.driver

uart_driver.sv

```
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);

    @(negedge uart_vif.clk);
    uart_vif.P_DATA     = stim_seq_item.P_DATA;
    uart_vif.rst_n      = stim_seq_item.rst_n;
    uart_vif.PAR_EN     = stim_seq_item.PAR_EN;
    uart_vif.PAR_TYP    = stim_seq_item.PAR_TYP;
    uart_vif.DATA_valid = 1'b1;
    @(posedge uart_vif.clk);
    uart_vif.DATA_valid = 1'b0;
    repeat (8) @(posedge uart_vif.clk);
    seq_item_port.item_done();

    `uvm_info("run_phase", stim_seq_item.convert2string_stimulus(), UVM_HIGH)
  end
endtask

endclass
endpackage
```

11.moniter

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");
        @(negedge uart_vif.clk);
        rsp_seq_item.P_DATA = uart_vif.P_DATA;
        rsp_seq_item.rst_n = uart_vif.rst_n;
        rsp_seq_item.DATA_valid = uart_vif.DATA_valid;
        rsp_seq_item.PAR_EN = uart_vif.PAR_EN;
        rsp_seq_item.PAR_TYP = uart_vif.PAR_TYP;
        rsp_seq_item.TX_OUT = uart_vif.TX_OUT;
        rsp_seq_item.busy = uart_vif.busy;
        mon_ap.write(rsp_seq_item);
        `uvm_info("run_phase" , rsp_seq_item.convert2string_stimulus(),UVM_HIGH)
    end
  endtask
endclass
endpackage
```

12.coverage

uart_coverage.sv

```
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 ;
       label_P_DATA : coverpoint seq_item_cov.P_DATA {
        bins data1 = {8'b00000000};
        bins data2[] = {8'b11111111 , 8'b00000000 , 8'b10101010};}

       label_PAR_TYP : coverpoint seq_item_cov.PAR_TYP;

       label_PAR_EN : coverpoint seq_item_cov.PAR_EN;

       label_DATA_VALID : coverpoint seq_item_cov.DATA_valid;

       label_tx_out : coverpoint seq_item_cov.TX_OUT;

       label_busy : coverpoint seq_item_cov.busy;

       cross_EN_TYP : cross  label_PAR_EN , label_PAR_TYP ;

       cross_DATA_valid : cross label_P_DATA , label_DATA_VALID
                          { ignore_bins bins_0_0 = binsof (label_P_DATA) intersect {0} &&
                                                   binsof (label_DATA_VALID) intersect {0};}

       cross_tx_out_busy : cross label_busy , label_tx_out
                          { ignore_bins bins_0_0 = binsof (label_busy) intersect {0} &&
                                                   binsof (label_tx_out) intersect {0};}

      endgroup
```

```
  function new (string name = "uart_cov" , 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
```

13.agent

uart_agent.sv

```
package uart_agent_pkg;
import uart_driver_pkg::*;
import uart_monitor_pkg::*;
import uart_sequencer_pkg::*;
import uart_seq_item_pkg::*;
import uart_conf_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 configration");

    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
```

14.env

uart_env.sv

```
package uart_env_pkg;
import uart_agent_pkg::*;
import uart_sco_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_sco 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_sco::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
```

15.test

uart_test.sv

```
package uart_test_pkg;
import uart_env_pkg::*;
import uart_conf_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_env" , 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);

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

    //main
    `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)

    //reset
    `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: uart_test
endpackage
```

16.top

```
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 GOLDEN_MODEL (uartif);
    bind uart uart_sva sv (uartif);

  initial begin
    uvm_config_db#(virtual uart_if)::set(null,"uvm_test_top","uart_IF",uartif);
    run_test("uart_test");
  end

endmodule
```

17.do file

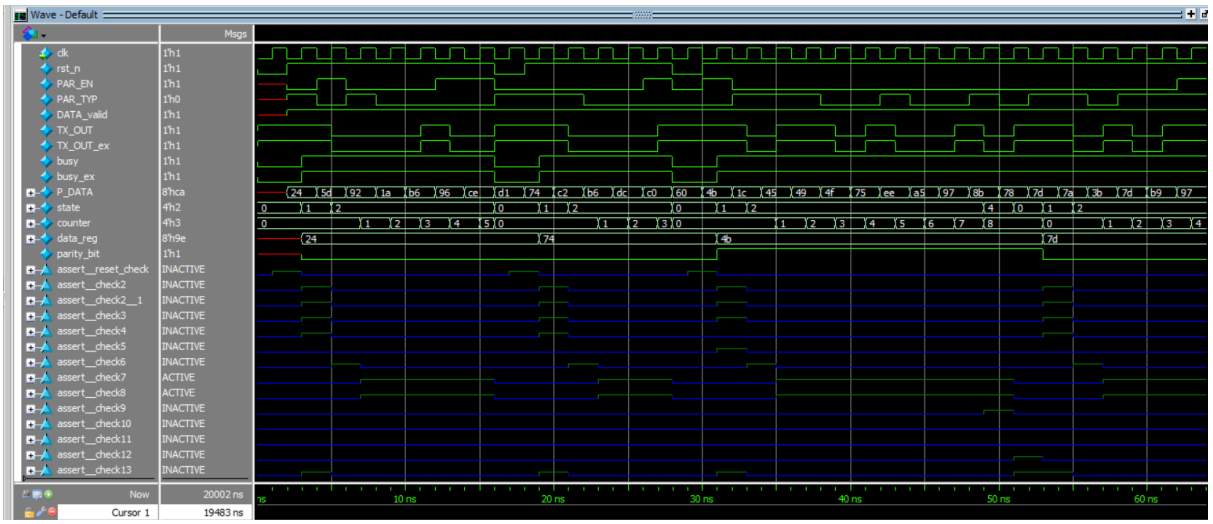
uart.do

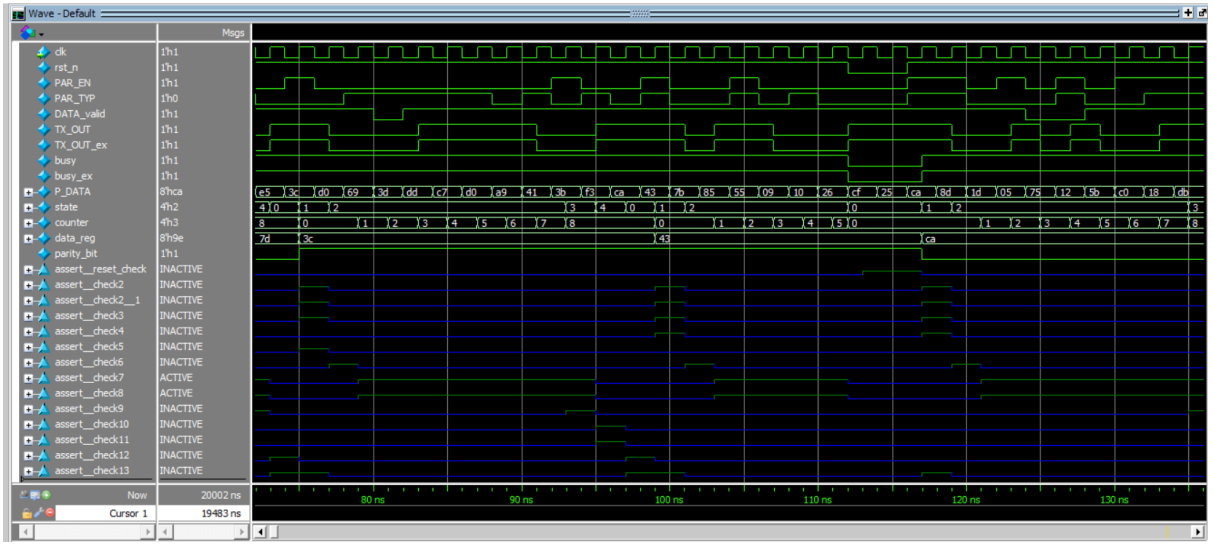
```
vlib work
vlog -f uart_files.list
vsim -voptargs=+acc work.uart_top -classdebug -uvmcontrol=all
add wave /uart_top/uartif/*
run -all
```

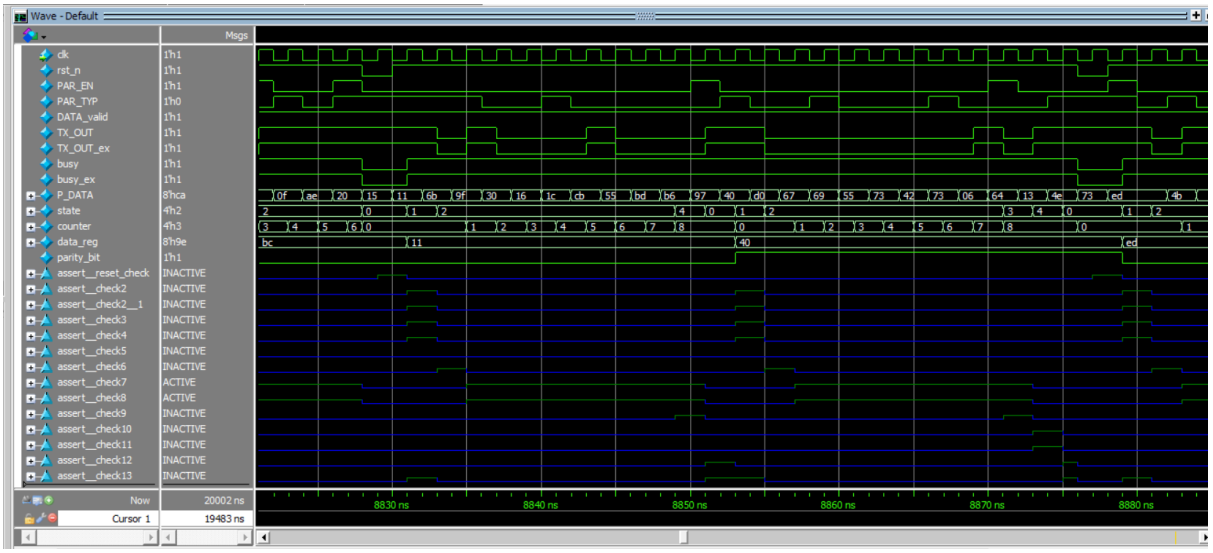
uart_files.list

```
uart.sv
uart_golden_model.sv
uart_sva.sv
uart_if.sv
uart_seq_item.sv
uart_scoreboard.sv
uart_config.sv
uart_seq.sv
uart_sequencer.sv
uart_driver.sv
uart_monitor.sv
uart_coverage.sv
uart_agent.sv
uart_env.sv
uart_test.sv
uart_top.sv
```

18.waveform


Wave - Default
Msgs
clk
rst_n
PAR_EN
PAR_TYP
DATA_valid
TX_OUT
TX_OUT_ex
busy
busy_ex
P_DATA
state
counter
data_reg
parity_bit
assert__reset_check
assert__check2
assert__check2__1
assert__check3
assert__check4
assert__check5
assert__check6
assert__check7
assert__check8
assert__check9
assert__check10
assert__check11
assert__check12
assert__check13
Now
20002 ns
Cursor 1
19483 ns
10 ns
20 ns
30 ns
40 ns
50 ns
60 ns


Wave - Default
Msgs
clk
rst_n
PAR_EN
PAR_TYP
DATA_valid
TX_OUT
TX_OUT_ex
busy
busy_ex
P_DATA
state
counter
data_reg
parity_bit
assert__reset_check
assert__check2
assert__check2__1
assert__check3
assert__check4
assert__check5
assert__check6
assert__check7
assert__check8
assert__check9
assert__check10
assert__check11
assert__check12
assert__check13
Now
20002 ns
Cursor 1
19483 ns
80 ns
90 ns
100 ns
110 ns
120 ns
130 ns


Wave - Default
Msgs
clk
rst_n
PAR_EN
PAR_TYP
DATA_valid
TX_OUT
TX_OUT_ex
busy
busy_ex
P_DATA
state
counter
data_reg
parity_bit
assert__reset_check
assert__check2
assert__check2__1
assert__check3
assert__check4
assert__check5
assert__check6
assert__check7
assert__check8
assert__check9
assert__check10
assert__check11
assert__check12
assert__check13
Now
20002 ns
Cursor 1
19483 ns
8830 ns
8840 ns
8850 ns
8860 ns
8870 ns
8880 ns

19.Assertion

/uart_seq_pkg::uart_main_seq::body/#ublk#2432689...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))
/uart_top/DUT/sv/assert__reset_check	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge uartif.clk) (~uar...
/uart_top/DUT/sv/assert__check2	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge uartif.clk) disab...
/uart_top/DUT/sv/assert__check2__1	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge uartif.clk) disab...
/uart_top/DUT/sv/assert__check3	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge uartif.clk) disab...
/uart_top/DUT/sv/assert__check4	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge uartif.clk) disab...
/uart_top/DUT/sv/assert__check5	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge uartif.clk) disab...
/uart_top/DUT/sv/assert__check6	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge uartif.clk) disab...
/uart_top/DUT/sv/assert__check7	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge uartif.clk) disab...
/uart_top/DUT/sv/assert__check8	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge uartif.clk) disab...
/uart_top/DUT/sv/assert__check9	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge uartif.clk) disab...
/uart_top/DUT/sv/assert__check10	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge uartif.clk) disab...
/uart_top/DUT/sv/assert__check11	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge uartif.clk) disab...
/uart_top/DUT/sv/assert__check12	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge uartif.clk) disab...
/uart_top/DUT/sv/assert__check13	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge uartif.clk) disab...

Cover Directives

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
/uart_top/DUT/sv/cover__check13	SVA	✓	Off	1712	1	Unlimited	1	100%		✓	0	0	0 ns	0
/uart_top/DUT/sv/cover__check12	SVA	✓	Off	692	1	Unlimited	1	100%		✓	0	0	0 ns	0
/uart_top/DUT/sv/cover__check11	SVA	✓	Off	177	1	Unlimited	1	100%		✓	0	0	0 ns	0
/uart_top/DUT/sv/cover__check10	SVA	✓	Off	177	1	Unlimited	1	100%		✓	0	0	0 ns	0
/uart_top/DUT/sv/cover__check9	SVA	✓	Off	716	1	Unlimited	1	100%		✓	0	0	0 ns	0
/uart_top/DUT/sv/cover__check8	SVA	✓	Off	6515	1	Unlimited	1	100%		✓	0	0	0 ns	0
/uart_top/DUT/sv/cover__check7	SVA	✓	Off	6515	1	Unlimited	1	100%		✓	0	0	0 ns	0
/uart_top/DUT/sv/cover__check6	SVA	✓	Off	946	1	Unlimited	1	100%		✓	0	0	0 ns	0
/uart_top/DUT/sv/cover__check5	SVA	✓	Off	493	1	Unlimited	1	100%		✓	0	0	0 ns	0
/uart_top/DUT/sv/cover__check4	SVA	✓	Off	481	1	Unlimited	1	100%		✓	0	0	0 ns	0
/uart_top/DUT/sv/cover__check3	SVA	✓	Off	974	1	Unlimited	1	100%		✓	0	0	0 ns	0
/uart_top/DUT/sv/cover__check2__1	SVA	✓	Off	974	1	Unlimited	1	100%		✓	0	0	0 ns	0
/uart_top/DUT/sv/cover__check2	SVA	✓	Off	974	1	Unlimited	1	100%		✓	0	0	0 ns	0
/uart_top/DUT/sv/cover__reset_check	SVA	✓	Off	329	1	Unlimited	1	100%		✓	0	0	0 ns	0

20.covergroup

Covergroups

Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_inst
/uart_coverage_pk...		100.00%					
TYPE cvr_gp		100.00%	100	100.00...		✓	
CVP cvr_g...		100.00%	100	100.00...		✓	
bin data...		1	1	100.00...		✓	
bin data...		1	1	100.00...		✓	
bin data...		33	1	100.00...		✓	
bin data...		40	1	100.00...		✓	
CVP cvr_g...		100.00%	100	100.00...		✓	
bin auto...		4947	1	100.00...		✓	
bin auto...		5052	1	100.00...		✓	
CVP cvr_g...		100.00%	100	100.00...		✓	
bin auto...		7461	1	100.00...		✓	
bin auto...		2538	1	100.00...		✓	
CVP cvr_g...		100.00%	100	100.00...		✓	
bin auto...		507	1	100.00...		✓	
bin auto...		9492	1	100.00...		✓	
CVP cvr_g...		100.00%	100	100.00...		✓	
bin auto...		4394	1	100.00...		✓	
bin auto...		5607	1	100.00...		✓	
CVP cvr_g...		100.00%	100	100.00...		✓	
bin auto...		379	1	100.00...		✓	
bin auto...		9622	1	100.00...		✓	
CROSS cvr...		100.00%	100	100.00...		✓	
bin <au...		1267	1	100.00...		✓	
bin <au...		3785	1	100.00...		✓	
bin <au...		1271	1	100.00...		✓	
bin <au...		3676	1	100.00...		✓	

CROSS cvr...	100.00%	100	100.00...
bin <au...	1267	1	100.00...
bin <au...	3785	1	100.00...
bin <au...	1271	1	100.00...
bin <au...	3676	1	100.00...
CROSS cvr...	100.00%	100	100.00...
bin <da...	39	1	100.00...
bin <da...	30	1	100.00...
bin <da...	1	1	100.00...
bin <da...	1	1	100.00...
bin <da...	1	1	100.00...
bin <da...	3	1	100.00...
ignore...	0	-	-
CROSS cvr...	100.00%	100	100.00...
bin <au...	5228	1	100.00...
bin <au...	4394	1	100.00...
bin <au...	379	1	100.00...
ignore...	0	-	-

21.output

Transcript

```
# ****************************************************************************
# UVM_INFO uart_test.sv(47) @ 2: uvm_test_top [run_phase] reset deasserted
# UVM_INFO uart_test.sv(50) @ 2: uvm_test_top [run_phase] stimulus generation started 1
# UVM_INFO uart_test.sv(52) @ 20000: uvm_test_top [run_phase] stimulus generation ended 1
# UVM_INFO uart_test.sv(55) @ 20000: uvm_test_top [run_phase] reset asserted
# UVM_INFO uart_test.sv(57) @ 20002: uvm_test_top [run_phase] reset deasserted
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 20002: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO uart_scoreboard.sv(44) @ 20002: uvm_test_top.env.sb [report_phase] corect =        10001
# UVM_INFO uart_scoreboard.sv(45) @ 20002: 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_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 20002 ns  Iteration: 61  Instance: /uart_top
# Break in Task uvm_pkg/uvm_root::run_test at C:/questasim64_2021.1/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh line 430

VSIM 17>
```