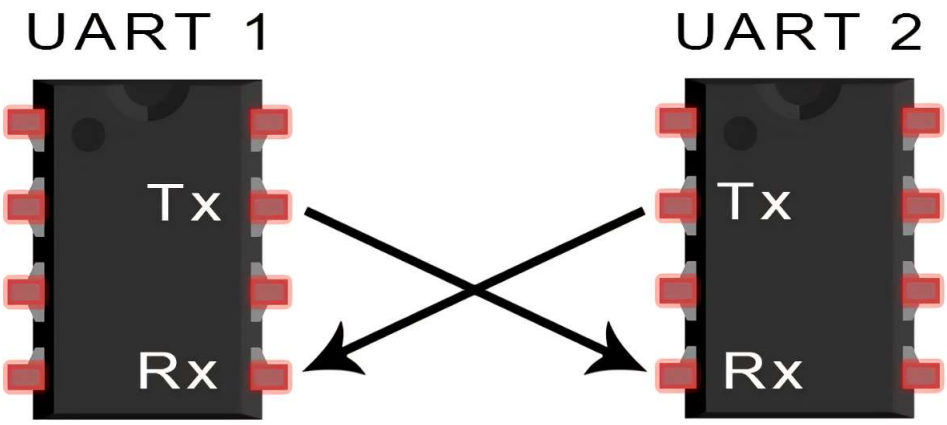
UART_TX

Verify by UVM

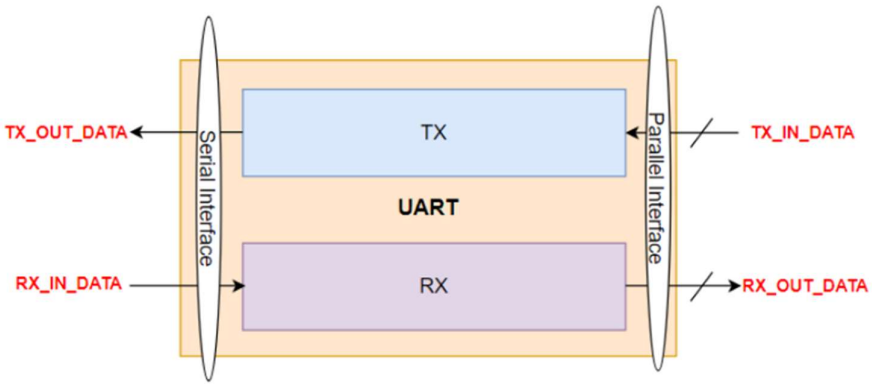
By: Mohammed Anwar
Abd allatif



Verification of UART Transmitter

Introduction :-

- There are many serial communication protocol as I2C, UART and SPI.
- A Universal Asynchronous Receiver/Transmitter (UART) is a block of circuitry responsible for implementing serial communication.
- UART is Full Duplex protocol (data transmission in both directions simultaneously)
- Transmitting UART converts parallel data from the master device (eg. CPU) into serial form and transmit in serial to receiving UART.
- Receiving UART will then convert the serial data back into parallel data for the receiving device.



RTL DESIGN_CODE:

D: > IEEE Verifiction > Final Project > UART_TX_UVM > UART_TX_files_UVM > UART_TX.sv

```
module uart_tx (uart_tx_if.DUT R_if);


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

  always @(posedge R_if.clk or negedge R_if.reset) begin
    if (!R_if.reset) begin
      state <= R_if.IDLE;
      R_if.TX_OUT <= 1'b1;
      R_if.Busy <= 1'b0;
      counter <= 0;
    end else begin
      case (state)
        R_if.IDLE: begin
          R_if.TX_OUT <= 1'b1;
          R_if.Busy <= 1'b0;
        counter <=0 ;
        if (R_if.DATA_VALID) begin
            data_reg <= R_if.P_DATA;
            parity_bit <= (R_if.PAR_TYP == 0 ) ? ~^R_if.P_DATA : ^R_if.P_DATA ;
            state <= R_if.START;
            R_if.Busy <= 1'b1;
        end
          end
        R_if.START: begin
         R_if.TX_OUT <= 1'b0;
         state <= R_if.DATA;
         counter <= 0 ;

        end

        R_if.DATA: begin
          R_if.TX_OUT <= data_reg[counter];
          counter <= counter + 1 ;
          if (counter == 7) begin
            state <= (R_if.PAR_EN) ? R_if.PARITY : R_if.STOP ;
            end
        end

        R_if.PARITY: begin
          R_if.TX_OUT <= parity_bit ;
          state <= R_if.STOP;
        end

        R_if.STOP: begin
          R_if.TX_OUT <= 1'b1;
          state <= R_if.IDLE;
        end
      endcase
    end
  end

```

GOLDEN MODEL_CODE :

D: > IEEE Verifiction > Final Project > UART_TX_UVM > UART_TX_files_UVM > UART_TX_GOLDEN.sv

```
module uart_tx_golden (uart_tx_if.GOLDEN R_if);


  reg [3:0] state_g = R_if.IDLE;
  reg [3:0] counter_g = 0;
  reg [7:0] data_reg_g;
  reg parity_bit_g;

  always @(posedge R_if.clk or negedge R_if.reset) begin
    if (!R_if.reset) begin
      state_g <= R_if.IDLE;
      R_if.TX_OUT_EXP <= 1'b1;
      R_if.Busy_exp <= 1'b0;
      counter_g <= 0;
    end else begin
      case (state_g)
        R_if.IDLE: begin
          R_if.TX_OUT_EXP <= 1'b1;
          R_if.Busy_exp <= 1'b0;
        counter_g <=0 ;
        if (R_if.DATA_VALID) begin
            data_reg_g <= R_if.P_DATA;
            parity_bit_g <= (R_if.PAR_TYP == 0 ) ? ~^R_if.P_DATA : ^R_if.P_DATA ;
            state_g <= R_if.START;
            R_if.Busy_exp <= 1'b1;
        end
          end
        R_if.START: begin
         R_if.TX_OUT_EXP <= 1'b0;
         state_g <= R_if.DATA;
         counter_g <= 0 ;
        end

        R_if.DATA: begin
          R_if.TX_OUT_EXP <= data_reg_g[counter_g];
          counter_g <= counter_g + 1 ;
          if (counter_g == 7) begin
            state_g <= (R_if.PAR_EN) ? R_if.PARITY : R_if.STOP ;
            end
        end

        R_if.PARITY: begin
          R_if.TX_OUT_EXP <= parity_bit_g ;
          state_g <= R_if.STOP;
        end

        R_if.STOP: begin
          R_if.TX_OUT_EXP <= 1'b1;
          state_g <= R_if.IDLE;
        end
      endcase
    end
  end

endmodule
```

Interface :

D: > IEEE Verifiction > Final Project > UART_TX_UVM > UART_TX_files_UVM > UART_TX_IF.sv

```
interface uart_tx_if(clk);

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

//---------------- input declaration ---------------
input bit clk;
logic reset ;
logic PAR_EN;
logic PAR_TYP;
logic  DATA_VALID;
logic [7:0] P_DATA;
//---------------- output declaration ---------------
logic TX_OUT , TX_OUT_EXP;
logic Busy , Busy_exp ;

//_____________________________ MODPORTS _____________________________________//

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

modport GOLDEN (input clk , reset , PAR_EN , PAR_TYP , DATA_VALID , P_DATA ,output TX_OUT_EXP , Busy_exp );









endinterface
```

Top module :

UART_TX_TOP.sv

```
module UART_TX_TOP ;
bit clk;
initial begin
clk = 0;
forever
#5 clk = ~clk ;

end

uart_tx_if R_if(clk);
uart_tx dut(R_if);
uart_tx_test tb(R_if);
bind uart_tx uart_tx_assertions T1(R_if);

endmodule
```

UART_TX TEST_pkg:

D: > IEEE Verifiction > Final Project > UART_TX_UVM > UART_TX_files_UVM > UART_TX_TEST_pkg.sv

```
package pack_test;
import pack_env::*;
import pack_config::*;
import pack_seqs::*;
import uvm_pkg::*;
`include "uvm_macros.svh"

class test extends uvm_test;
  `uvm_component_utils(test);

  virtual uart_tx_if uart_vif;
  uart_config uart_cfg;
  uart_env env;
  uart_reset_sequence                  seq_one;
  uart_tx_noparity_sequence            seq_two;
  uart_tx_parity_sequence              seq_three;
  uart_tx_random_sequence              seq_four;
  uart_tx_coverage_sequence            seq_five;


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

  // Build phase
  function void build_phase(uvm_phase phase);
    super.build_phase(phase);

    uart_cfg = uart_config::type_id::create("uart_cfg");
    env      = uart_env::type_id::create("env", this);
    seq_one = uart_reset_sequence::type_id::create("seq_one");
    seq_two = uart_tx_noparity_sequence::type_id::create("seq_two");
    seq_three = uart_tx_parity_sequence::type_id::create("seq_three");
    seq_four = uart_tx_random_sequence::type_id::create("seq_four");
    seq_five = uart_tx_coverage_sequence::type_id::create("seq_five");


    uvm_config_db#(uart_config)::set(this, "*", "CGO", uart_cfg);

    if (!uvm_config_db#(virtual uart_tx_if)::get(this , "" , "UART_LL" , uart_cfg.uart_vif)) begin
      `uvm_fatal("build_phase" , "Unable to get virtual interface")
    end
  endfunction
```

```
  // Run phase
  task run_phase(uvm_phase phase);
    super.run_phase(phase);
    phase.raise_objection(this);

    `uvm_info("run_phase", "Reset sequence started", UVM_LOW);
    seq_one.start(env.agt.sqr);
    `uvm_info("run_phase", "Reset sequence finished", UVM_LOW);

    `uvm_info("run_phase", "noparity_sequence  started", UVM_LOW);
    seq_two.start(env.agt.sqr);
    `uvm_info("run_phase", "noparity_sequence  finished", UVM_LOW);

    `uvm_info("run_phase", "tx_parity_sequence  started", UVM_LOW);
    seq_three.start(env.agt.sqr);
    `uvm_info("run_phase", "tx_parity_sequence  finished", UVM_LOW);


    `uvm_info("run_phase", "Random  sequence started", UVM_LOW);
    seq_four.start(env.agt.sqr);
    `uvm_info("run_phase", "Random  sequence finished", UVM_LOW);


    `uvm_info("run_phase", "Random  sequence started", UVM_LOW);
    seq_five.start(env.agt.sqr);
    `uvm_info("run_phase", "Random  sequence finished", UVM_LOW);









    phase.drop_objection(this);
  endtask

endclass
endpackage
```

UART_TX ENVIRONMENT_pkg:

D: > IEEE Verifiction > Final Project > UART_TX_UVM > UART_TX_files_UVM > UART_TX_ENVIRONMENT_pkg.sv

```
1   package pack_env;
2   import pack_seq_item::*;
3   import pack_agent::*;
4   import pack_coverage::*;
5   import scoreboard::*;
6   import uvm_pkg::*;
7   `include "uvm_macros.svh"
8   class uart_env extends uvm_env;
9   `uvm_component_utils(uart_env);
10  uart_agent agt;
11  uart_scoreboard sb;
12  uart_coverage cov;
13  uvm_analysis_port #(shift_reg_seq_item)agt_ap;
14
15      function new(string name="uart_env" , uvm_component parent = null);
16      super.new(name,parent);
17          endfunction
18  /************** BUILD PHASE **********************/
19          function void build_phase(uvm_phase phase);
20          super.build_phase(phase);
21      agt=uart_agent::type_id::create("agt",this);
22      sb=uart_scoreboard::type_id::create("sb",this);
23      cov=uart_coverage::type_id::create("cov",this);
24      agt_ap=new("agt_ap",this);
25          endfunction
26  /****************** CONNECTED PHASE  *************/
27   function void connect_phase(uvm_phase phase);
28              super.connect_phase(phase);
29              agt.agt_ap.connect(sb.sb_export);
30              agt.agt_ap.connect (cov.cov_export);
31          endfunction
32
33  endclass
34
35  endpackage
```

UART_TX OBJECT_pkg:

D: > IEEE Verifiction > Final Project > UART_TX_UVM > UART_TX_files_UVM > UART_TX_GONFIGRATION_pkg.sv

```
1   package pack_config;
2   import uvm_pkg::*;
3   `include "uvm_macros.svh"
4   class uart_config extends uvm_object;
5   `uvm_object_utils(uart_config);
6   virtual uart_tx_if uart_vif;
7
8       function new(string name = "uart_config");
9       super.new(name);
10
11      endfunction
12  endclass
13
14
15  endpackage
```

UART_TX SEQ_ITEM_pkg:

D: > IEEE Verifiction > Final Project > UART_TX_UVM > UART_TX_files_UVM > UART_TX_SEQ_ITEM_pkg.sv

```
package pack_seq_item;

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

class shift_reg_seq_item extends uvm_sequence_item;

  // FSM state encoding (for future use or ref)
  parameter IDLE   = 0;
  parameter START  = 1;
  parameter DATA   = 2;
  parameter PARITY = 3;
  parameter STOP   = 4;

  `uvm_object_utils(shift_reg_seq_item)

  // Control signals
  rand logic reset;                      // Active-low reset
  rand logic PAR_EN;                     // 0: disable parity, 1: enable parity
  rand logic PAR_TYP;                    // 0: even, 1: odd
  rand logic DATA_VALID;
  logic clk;

  // Data
  rand logic [7:0] P_DATA;

  // DUT outputs
  logic TX_OUT, TX_OUT_EXP;
  logic Busy, Busy_exp;

  // Constructor
  function new(string name = "shift_reg_seq_item");
    super.new(name);
  endfunction
```

```
  ////////////////////////////////////////////////////////////////////////////////////////
  // Constraints - Functional Coverage Related
  ////////////////////////////////////////////////////////////////////////////////////////

  // I- Reset active for ~3% of simulation time
  constraint c_reset_active {
    reset dist {0 := 3, 1 := 97}; // Active-low reset
  }

  // II- PAR_TYP toggle: 50% even, 50% odd
  constraint c_par_typ_toggle {
    PAR_TYP dist {0 := 50, 1 := 50};
  }

  // III- PAR_EN enabled 25% of time
  constraint c_par_en {
    PAR_EN dist {1 := 25, 0 := 75};
  }

  // IV- DATA_VALID active most of simulation time
  constraint c_data_valid {
    DATA_VALID dist {1 := 85, 0 := 15};
  }

  // V- P_DATA specs:
  // a. LSB = 1 in 80% of cases
  constraint c_lsb_one {
    (P_DATA[0] == 1) dist {1 := 80, 0 := 20};
  }

  // b. Specific patterns (11111111, 00000000, 10101010) appear 4% of time
  constraint c_specific_patterns {
    P_DATA dist {
      8'b11111111 := 1,
      8'b00000000 := 1,
      8'b10101010 := 1,
      [8'b00000001:8'b11111110] := 96
    };
  }
```

```
  ////////////////////////////////////////////////////////////////////////////
  // Utility methods
  ////////////////////////////////////////////////////////////////////////////

  function string convert2string();
    return $sformatf(" %s :reset=%0b, PAR_EN=%0b, PAR_TYP=%0b, DATA_VALID=%0b, P_DATA=0x%0h, TX_OUT=%0b, Busy=%0b , TX_OUT_EXP=%0b , Busy_exp=%0b ",
      super.convert2string() ,reset, PAR_EN, PAR_TYP, DATA_VALID, P_DATA, TX_OUT, Busy , TX_OUT_EXP , Busy_exp );
  endfunction

  function string convert2string_stimulus();
    return $sformatf("%s reset=%0b, PAR_EN=%0b, PAR_TYP=%0b, DATA_VALID=%0b, P_DATA=%0h",
      super.convert2string(),  reset, PAR_EN, PAR_TYP, DATA_VALID, P_DATA);
  endfunction

endclass

endpackage
```

UART_TX SEQUNCER_PKG :

D: > IEEE Verifiction > Final Project > UART_TX_UVM > UART_TX_files_UVM > UART_TX_SEQUENCER_pkg.sv

```
1   package pack_sequencer;
2     import uvm_pkg::*;
3     `include "uvm_macros.svh"
4     import pack_seq_item::*;
5
6     class uart_sequencer extends uvm_sequencer #(shift_reg_seq_item);// calss name of seq_item
7       `uvm_component_utils(uart_sequencer);
8
9       function new(string name = "uart_sequencer", uvm_component parent = null);
10        super.new(name, parent);
11      endfunction
12    endclass
13  endpackage
14
```

UART_TX SEQUNCES_PKG :

D: > IEEE Verifiction > Final Project > UART_TX_UVM > UART_TX_files_UVM > UART_TX_SEQS_pkg.sv

```
1   package pack_seqs;
2
3     import uvm_pkg::*;
4     `include "uvm_macros.svh"
5     import pack_seq_item::*;
6
7     ////////////////////////////////////////////////////////////////////////////
8     // Sequence 1: Reset Sequence (active-low)
9     ////////////////////////////////////////////////////////////////////////////
10    class uart_reset_sequence extends uvm_sequence #(shift_reg_seq_item);
11      `uvm_object_utils(uart_reset_sequence)
12      shift_reg_seq_item seq_item;
13
14      function new(string name = "uart_reset_sequence");
15        super.new(name);
16      endfunction
17
18      task body();
19        seq_item = shift_reg_seq_item::type_id::create("seq_item");
20        start_item(seq_item);
21        seq_item.reset = 0; // active-low
22        seq_item.P_DATA = 8'b0;
23        seq_item.PAR_EN = 0;
24        seq_item.PAR_TYP = 0;
25        finish_item(seq_item);
26      endtask
27    endclass
28
29    ////////////////////////////////////////////////////////////////////////////
30    // Sequence 2: Send data with parity disabled
31    ////////////////////////////////////////////////////////////////////////////
32    class uart_tx_noparity_sequence extends uvm_sequence #(shift_reg_seq_item);
33      `uvm_object_utils(uart_tx_noparity_sequence)
34      shift_reg_seq_item seq_item;
35
36      function new(string name = "uart_tx_noparity_sequence");
37        super.new(name);
38      endfunction
39
40      task body();
41        seq_item = shift_reg_seq_item::type_id::create("seq_item");
42        start_item(seq_item);
43        seq_item.reset       = 1;
44        seq_item.DATA_VALID  = 1;
45        seq_item.PAR_EN      = 0;
46        seq_item.PAR_TYP     = 0;
47        seq_item.P_DATA      = 8'hA5;
48        finish_item(seq_item);
49      endtask
50    endclass
51
```

```
////////////////////////////////////////////////////////////////////////////////
// Sequence 3: Send data with parity enabled (even or odd)
////////////////////////////////////////////////////////////////////////////////
class uart_tx_parity_sequence extends uvm_sequence #(shift_reg_seq_item);
  `uvm_object_utils(uart_tx_parity_sequence)
  shift_reg_seq_item seq_item;

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

  task body();
    seq_item = shift_reg_seq_item::type_id::create("seq_item");
    start_item(seq_item);
    seq_item.reset       = 1;
    seq_item.DATA_VALID  = 1;
    seq_item.PAR_EN      = 1;
    seq_item.PAR_TYP     = $urandom_range(0,1); // even or odd
    seq_item.P_DATA      = 8'h3C;
    finish_item(seq_item);
  endtask
endclass

////////////////////////////////////////////////////////////////////////////////
// Sequence 4: Send random legal stimulus (fully randomized)
////////////////////////////////////////////////////////////////////////////////
class uart_tx_random_sequence extends uvm_sequence #(shift_reg_seq_item);
  `uvm_object_utils(uart_tx_random_sequence)
  shift_reg_seq_item seq_item;

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

  task body();
    repeat (999) begin
      seq_item = shift_reg_seq_item::type_id::create("seq_item");
      start_item(seq_item);
      assert(seq_item.randomize() with {
        DATA_VALID == 1'b0;
      });
      finish_item(seq_item);
    end
  endtask
endclass
```

```
////////////////////////////////////////////////////////////////////////////////
// Sequence 5: Coverage-oriented sequence
////////////////////////////////////////////////////////////////////////////////
class uart_tx_coverage_sequence extends uvm_sequence #(shift_reg_seq_item);
  `uvm_object_utils(uart_tx_coverage_sequence)
  shift_reg_seq_item seq_item;

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

  task body();
    int data_val;
    int valid;
    int par_typ;

    // Test all data values from 0 to 7 with DATA_VALID = 0 and 1, PAR_EN = 0
    for (data_val = 0; data_val <= 7; data_val++) begin
      for (valid = 0; valid <= 1; valid++) begin
        seq_item = shift_reg_seq_item::type_id::create("seq_item");
        start_item(seq_item);
        seq_item.reset        = 1;
        seq_item.DATA_VALID   = valid;
        seq_item.PAR_EN       = 0;
        seq_item.PAR_TYP      = 0;
        seq_item.P_DATA       = data_val;
        finish_item(seq_item);
      end
    end

    // Check both parity types with PAR_EN = 1
    for (par_typ = 0; par_typ <= 1; par_typ++) begin
      seq_item = shift_reg_seq_item::type_id::create("seq_item");
      start_item(seq_item);
      seq_item.reset        = 1;
      seq_item.DATA_VALID   = 1;
      seq_item.PAR_EN       = 1;
      seq_item.PAR_TYP      = par_typ;
      seq_item.P_DATA       = 5;
      finish_item(seq_item);
    end

    // Generate additional randomized valid stimuli for coverage
    repeat (200) begin
      seq_item = shift_reg_seq_item::type_id::create("seq_item");
      start_item(seq_item);
      assert(seq_item.randomize() with {
        P_DATA inside {[0:7]};
        DATA_VALID == 1;
        PAR_EN inside {0,1};
        PAR_TYP inside {0,1};
      });
      finish_item(seq_item);
    end
  endtask
endclass

endpackage
```

UART_TX DRIVER_PKG :

```
package pack_driver;
import pack_config::*;
import uvm_pkg::*;
import pack_seq_item::*;
`include "uvm_macros.svh"
class uart_driver extends uvm_driver #(shift_reg_seq_item);
`uvm_component_utils(uart_driver);
//ram_config ram_cfg;
virtual uart_tx_if sh_vif;
shift_reg_seq_item stim_seq_item;
/********** COSTRACTOR ************/
    function new(string name ="uart_driver", uvm_component parent = null);
    super.new(name,parent);
    endfunction
    //***************RUN_PHASE*****************//

task run_phase(uvm_phase phase);
    super.run_phase(phase);
    forever
     begin
      // Create a new sequence item
      stim_seq_item = shift_reg_seq_item::type_id::create("stim_seq_item");

      // Get transaction from sequencer
      seq_item_port.get_next_item(stim_seq_item);

      // Drive stimulus to DUT interface
      sh_vif.reset         = stim_seq_item.reset;
      sh_vif.PAR_EN         = stim_seq_item.PAR_EN;
      sh_vif.PAR_TYP        = stim_seq_item.PAR_TYP;
      sh_vif.P_DATA       = stim_seq_item.P_DATA;
       sh_vif.DATA_VALID      = stim_seq_item.DATA_VALID;

      // Wait for one negative clock edge
      @(negedge sh_vif.clk);

      // Capture DUT output
      stim_seq_item.TX_OUT = sh_vif.TX_OUT;
       stim_seq_item.Busy = sh_vif.Busy;


      // Complete transaction
      seq_item_port.item_done();

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

endclass
endpackage
```

UART_TX MONITOR_PKG :

D: > IEEE Verification > Final Project > UART_TX_UVM > UART_TX_files_UVM > UART_TX_MONITOR_pkg.sv

```
package pack_mon;

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

  class uart_monitor extends uvm_monitor;
    `uvm_component_utils(uart_monitor)

    virtual uart_tx_if sh_vif;
    shift_reg_seq_item rsp_seq_item;
    uvm_analysis_port #(shift_reg_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 = shift_reg_seq_item::type_id::create("rsp_seq_item");

        @(negedge sh_vif.clk);

        rsp_seq_item.reset    = sh_vif.reset;
        rsp_seq_item.PAR_TYP = sh_vif.PAR_TYP;
        rsp_seq_item.PAR_EN = sh_vif.PAR_EN;
        rsp_seq_item.DATA_VALID    = sh_vif.DATA_VALID;
        rsp_seq_item.P_DATA    = sh_vif.P_DATA;
        rsp_seq_item.TX_OUT      = sh_vif.TX_OUT;
        rsp_seq_item.Busy     = sh_vif.Busy;
        rsp_seq_item.TX_OUT_EXP      = sh_vif.TX_OUT_EXP;
        rsp_seq_item.Busy_exp     = sh_vif.Busy_exp;

        mon_ap.write(rsp_seq_item);

        `uvm_info("run_phase", rsp_seq_item.convert2string(), UVM_HIGH);
      end
    endtask

  endclass

endpackage
```

UART_TX AGENT_PKG :

D: > IEEE Verifiction > Final Project > UART_TX_UVM > UART_TX_files_UVM > UART_TX_AGENT_pkg.sv

```
package pack_agent;
  `include "uvm_macros.svh"
  import uvm_pkg::*;
  import pack_driver::*;
  import pack_sequencer::*;
  import pack_seq_item::*;
  import pack_mon::*;
  import pack_config::*;
  import pack_seqs::*;

  class uart_agent extends uvm_agent;
    `uvm_component_utils(uart_agent);

    uart_config uart_config_obj_driver;   // ✅ Correct type
    uart_driver uart_dr;
    uart_sequencer sqr;
    uart_monitor mon;
    uvm_analysis_port #(shift_reg_seq_item) agt_ap;

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

    function void build_phase(uvm_phase phase);
      super.build_phase(phase);

      if (!uvm_config_db #(uart_config)::get(this, "", "CGO", uart_config_obj_driver)) begin
        `uvm_fatal("build_phase", "DRIVER - unable to get the virtual interface");
      end

      uart_dr = uart_driver::type_id::create("uart_dr", this);
      sqr = uart_sequencer::type_id::create("sqr", this);
      mon = uart_monitor::type_id::create("mon", this);
      agt_ap = new("agt_ap", this);
    endfunction

    function void connect_phase(uvm_phase phase);
      super.connect_phase(phase);


        uart_dr.seq_item_port.connect(sqr.seq_item_export);
        uart_dr.sh_vif = uart_config_obj_driver.uart_vif;


      mon.sh_vif = uart_config_obj_driver.uart_vif;
      mon.mon_ap.connect(agt_ap);
    endfunction
  endclass
endpackage

```

UART_TX COVERAGE_PKG :

D: > IEEE Verifiction > Final Project > UART_TX_UVM > UART_TX_files_UVM > UART_TX_COVERAGE_pkg.sv

```
package pack_coverage;

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

  class uart_coverage extends uvm_component;
    `uvm_component_utils(uart_coverage)

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

    localparam int MEM_DEPTH = 256;
    localparam int ADDR_SIZE = $clog2(MEM_DEPTH);
    localparam int MEM_WIDTH = 8;

   covergroup UART_CG;

  cp_PAR_EN    : coverpoint seq_item_cov.PAR_EN;
  cp_PAR_TYPE  : coverpoint seq_item_cov.PAR_TYP;
  cp_P_VALID   : coverpoint seq_item_cov.DATA_VALID;
  cp_P_DATA    : coverpoint seq_item_cov.P_DATA {
    bins low_vals[] = {[0:7]}; // smaller set
    ignore_bins u = {[8:255]};
  }

  // Cross coverpoints
  cross_PAR_EN_PAR_TYPE : cross cp_PAR_EN, cp_PAR_TYPE {
    ignore_bins invalid = binsof(cp_PAR_EN) intersect {0} &&
                          binsof(cp_PAR_TYPE) intersect {1};
  }

  cross_P_DATA_P_VALID  : cross cp_P_DATA, cp_P_VALID;
  cross_TX_OUT_BUSY     : cross seq_item_cov.TX_OUT, seq_item_cov.Busy;

endgroup

```

```
  function new(string name = " uart_coverage", uvm_component parent = null);
    super.new(name, parent);
    UART_CG= 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);
      UART_CG.sample();
    end
  endtask

endclass
endpackage
```

UART_TX SCOREBOARD_PKG :

D: > IEEE Verifiction > Final Project > UART_TX_UVM > UART_TX_files_UVM > UART_TX_SCOREBOARD_pkg.sv

```
package scoreboard;
  `include "uvm_macros.svh"
  import uvm_pkg::*;
  import pack_seq_item::*;  // Using RAM-specific sequence item

  class uart_scoreboard extends uvm_scoreboard;
    `uvm_component_utils(uart_scoreboard)

    // Analysis ports and FIFO
    uvm_analysis_export #(shift_reg_seq_item) sb_export;
    uvm_tlm_analysis_fifo #(shift_reg_seq_item) sb_fifo;
    shift_reg_seq_item seq_item_sb;


    // Statistics
    int error_count = 0;
    int correct_count = 0;
    // CONSTRACTOR
    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_EXP) && (seq_item_sb.Busy != seq_item_sb.Busy_exp)) begin
          `uvm_error("run_phase",$sformatf("compartion failled at referance = %0d and %0d", seq_item_sb.TX_OUT_EXP , seq_item_sb.Busy_exp))
          error_count++;
        end
        else
        correct_count++;
      end
    endtask

    function void report_phase(uvm_phase phase);
      super.report_phase(phase);
      `uvm_info("SB_REPORT",
        $sformatf("\n--- UART_TX Scoreboard Report ---\n" +
                  "Mismatches:       %0d\n" +
                  "Match             %0d\n" +
                  "----------------------------",
                  error_count,
                  correct_count),
        UVM_LOW)
    endfunction
  endclass
endpackage
```

UART_TX ASSERTIONS:

D: > IEEE Verifiction > Final Project > UART_TX_UVM > UART_TX_files_UVM > UART_TX_SVA.sv

```
module uart_tx_assertions (uart_tx_if.DUT vif);

// Final reset checks after reset is deasserted
always_comb begin
  if (!vif.reset) begin
    a_assert: assert final(uart_tx.state == vif.IDLE);
    b_assert: assert final(vif.TX_OUT == 1'b1);
    c_assert: assert final(vif.Busy == 1'b0);
    d_assert: assert final(uart_tx.counter == 0);
  end
end

// 1. TX_OUT should be high and Busy low in IDLE state
property idle_behavior;
  @(posedge vif.clk) disable iff (!vif.reset)
  (uart_tx.state == vif.IDLE && vif.DATA_VALID == 1'b0) |=>
  (vif.TX_OUT == 1'b1 && vif.Busy == 1'b0 && uart_tx.counter == 0);
endproperty
p1: assert property(idle_behavior) else $error("TX_OUT should be 1 and Busy 0 in IDLE state.");
c1: cover property(idle_behavior);

// 2. FSM should remain in IDLE when DATA_VALID = 0
property idle_2;
  @(posedge vif.clk) disable iff (!vif.reset)
  (uart_tx.state == vif.IDLE && vif.DATA_VALID == 1'b0) |=> (uart_tx.state == vif.IDLE);
endproperty
p2: assert property(idle_2) else $error("State should remain IDLE when DATA_VALID is 0.");
c2: cover property(idle_2);

// 3. START state: TX_OUT should be 0 and counter = 0
property start_bit_low;
  @(posedge vif.clk) disable iff (!vif.reset)
  (uart_tx.state == vif.START) |=> (vif.TX_OUT == 1'b0 && uart_tx.counter == 0);
endproperty
p3: assert property(start_bit_low) else $error("TX_OUT should be 0 and counter 0 in START state.");
c3: cover property(start_bit_low);

// 4. DATA state continues when counter != 7
property data_bit_1;
  @(posedge vif.clk) disable iff (!vif.reset)
  (uart_tx.state == vif.DATA && uart_tx.counter != 7) |=> (uart_tx.state == vif.DATA);
endproperty
p4: assert property(data_bit_1) else $error("FSM should stay in DATA state while counter != 7.");
c4: cover property(data_bit_1);

// 5. TX_OUT should match data_reg[counter] and counter must increment
property data_bit_match;
  @(posedge vif.clk) disable iff (!vif.reset)
  (uart_tx.state == vif.DATA && uart_tx.counter != 7) |=>
  (vif.TX_OUT == uart_tx.data_reg[$past(uart_tx.counter)] &&
   uart_tx.counter == $past(uart_tx.counter) + 1);
endproperty
p5: assert property(data_bit_match) else $error("TX_OUT mismatch or counter didn't increment in DATA state.");
c5: cover property(data_bit_match);

// 6. PARITY state: TX_OUT should match calculated parity bit
property parity_correct;
  @(posedge vif.clk) disable iff (!vif.reset)
  (uart_tx.state == vif.PARITY) |=> (vif.TX_OUT == uart_tx.parity_bit);
endproperty
p6: assert property(parity_correct) else $error("TX_OUT does not match expected parity bit.");
c6: cover property(parity_correct);

// 7. STOP state: TX_OUT should be high
property stop_bit_high;
  @(posedge vif.clk) disable iff (!vif.reset)
  (uart_tx.state == vif.STOP) |=> (vif.TX_OUT == 1'b1);
endproperty
p7: assert property(stop_bit_high) else $error("TX_OUT must be high in STOP state.");
c7: cover property(stop_bit_high);
```

```
// 8. No transition from IDLE if DATA_VALID is not asserted
property no_start_without_valid;
  @(posedge vif.clk) disable iff (!vif.reset)
  (uart_tx.state == vif.IDLE && vif.DATA_VALID == 0) |=> (uart_tx.state == vif.IDLE);
endproperty
p8: assert property(no_start_without_valid) else $error("FSM left IDLE without DATA_VALID = 1.");
c8: cover property(no_start_without_valid);

// 9. Even parity logic from IDLE
property idle_even_parity;
  @(posedge vif.clk) disable iff (!vif.reset)
  (uart_tx.state == vif.IDLE && vif.DATA_VALID == 1 && vif.PAR_TYP == 0) |->
  (uart_tx.parity_bit == ~^$past(vif.P_DATA) &&
   vif.Busy == 1'b1 &&
   uart_tx.data_reg == $past(vif.P_DATA));
endproperty
p9: assert property(idle_even_parity) else $error("Even parity incorrect after IDLE.");
c9: cover property(idle_even_parity);

// 10. Odd parity logic from IDLE
property idle_odd_parity;
  @(posedge vif.clk) disable iff (!vif.reset)
  (uart_tx.state == vif.IDLE && vif.DATA_VALID == 1 && vif.PAR_TYP == 1) |->
  (uart_tx.parity_bit == ^$past(vif.P_DATA) &&
   vif.Busy == 1'b1 &&
   uart_tx.data_reg == $past(vif.P_DATA));
endproperty
p10: assert property(idle_odd_parity) else $error("Odd parity incorrect after IDLE.");
c10: cover property(idle_odd_parity);

// 11. Transition from IDLE to START when DATA_VALID = 1
property idle_to_start;
  @(posedge vif.clk) disable iff (!vif.reset)
  (uart_tx.state == vif.IDLE && vif.DATA_VALID == 1) |=> (uart_tx.state == vif.START);
endproperty
p11: assert property(idle_to_start) else $error("Should move from IDLE to START when DATA_VALID = 1.");
c11: cover property(idle_to_start);

// 12. DATA to PARITY transition when counter==7 and PAR_EN==1
property data_to_parity;
  @(posedge vif.clk) disable iff (!vif.reset)
  (uart_tx.state == vif.DATA && uart_tx.counter == 7 && vif.PAR_EN == 1) |->
  (uart_tx.state == vif.PARITY);
endproperty
p12: assert property(data_to_parity) else $error("Should go to PARITY after DATA if PAR_EN == 1.");
c12: cover property(data_to_parity);

// 13. DATA to STOP transition when counter==7 and PAR_EN==0
property data_to_stop;
  @(posedge vif.clk) disable iff (!vif.reset)
  (uart_tx.state == vif.DATA && uart_tx.counter == 7 && vif.PAR_EN == 0) |->
  (uart_tx.state == vif.STOP);
endproperty
p13: assert property(data_to_stop) else $error("Should go to STOP after DATA if PAR_EN == 0.");
c13: cover property(data_to_stop);

// 14. PARITY to STOP transition
property parity_to_stop;
  @(posedge vif.clk) disable iff (!vif.reset)
  (uart_tx.state == vif.PARITY) |->
  (vif.TX_OUT == $past(uart_tx.parity_bit) && uart_tx.state == vif.STOP);
endproperty
p14: assert property(parity_to_stop) else $error("Should go to STOP after PARITY.");
c14: cover property(parity_to_stop);

// 15. STOP to IDLE transition
property stop_to_idle;
  @(posedge vif.clk) disable iff (!vif.reset)
  (uart_tx.state == vif.STOP) |->
  (vif.TX_OUT == 1 && uart_tx.state == vif.IDLE);
endproperty
p15: assert property(stop_to_idle) else $error("Should go to IDLE after STOP.");
c15: cover property(stop_to_idle);

endmodule
```

DO_File :

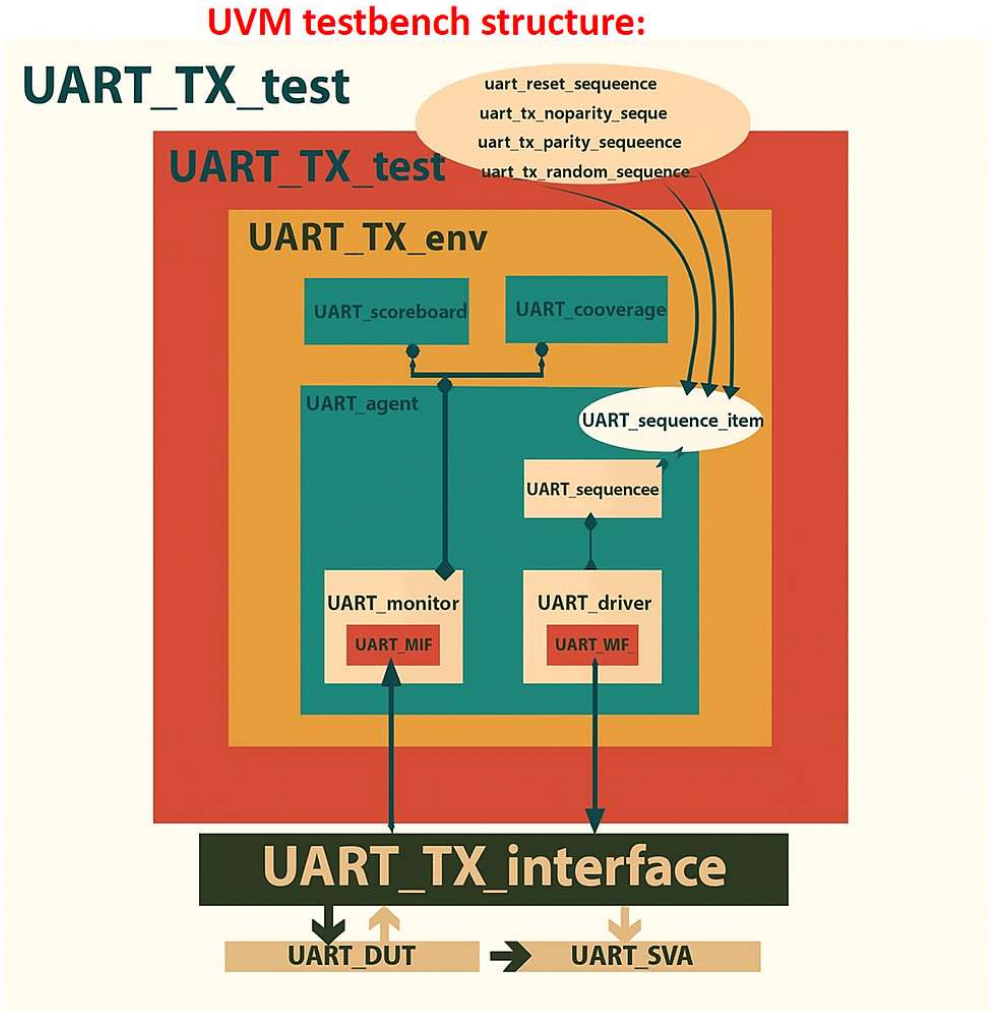
D: > IEEE Verifiction > Final Project > UART_TX_UVM > UART_TX_files_UVM > run.do

```
vlog -sv +cover +acc \
UART_TX_IF.sv \
UART_TX.sv \
UART_TX_GOLDEN.sv \
UART_TX_SVA.sv \
UART_TX_SEQ_ITEM_pkg.sv \
UART_TX_SEQS_pkg.sv \
UART_TX_COVERAGE_pkg.sv \
UART_TX_DRIVER_pkg.sv \
UART_TX_MONITOR_pkg.sv \
UART_TX_SCOREBOARD_pkg.sv \
UART_TX_SEQUENCER_pkg.sv \
UART_TX_AGENT_pkg.sv \
UART_TX_GONFIGRATION_pkg.sv \
UART_TX_ENVIRONMENT_pkg.sv \
UART_TX_TEST_pkg.sv \
UART_TX_TOP.sv \
+define+UVM_NO_DPI

vsim -voptargs=+acc work.uart_top -classdebug -uvmcontrol=all -coverage

add wave /uart_top/clk
add wave /uart_top/R_if/*

coverage save uart_tx.ucdb -onexit
run -all

vcover report uart_tx.ucdb -details -all -annotate -output coverage.txt

```

List:

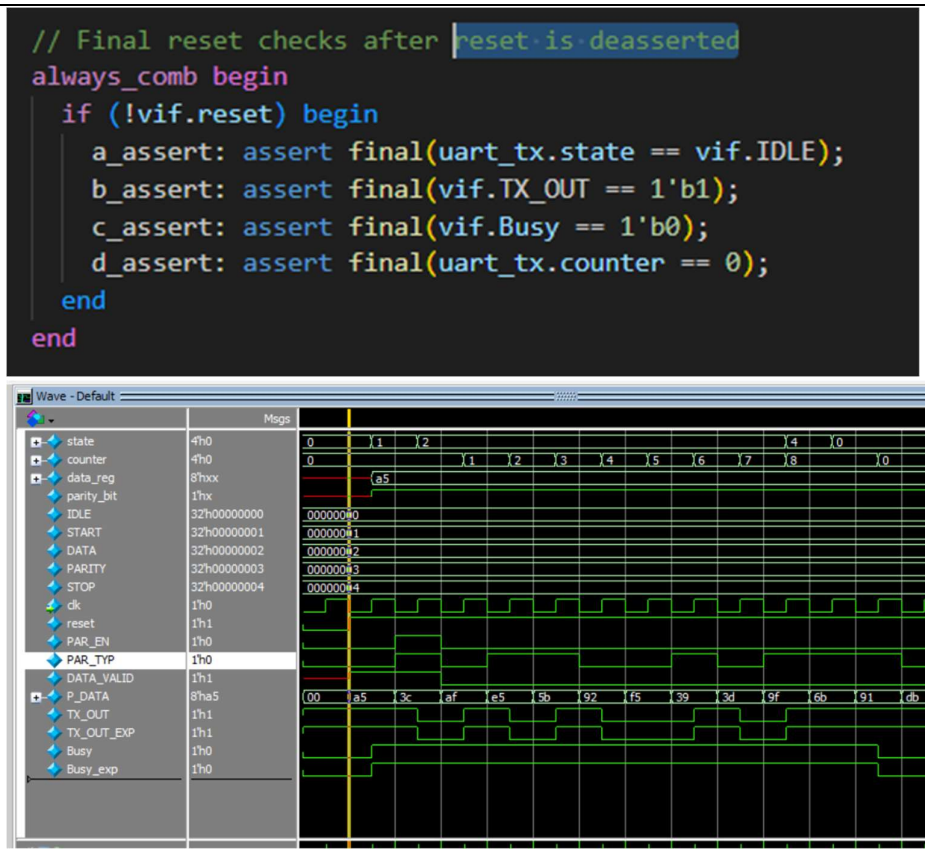
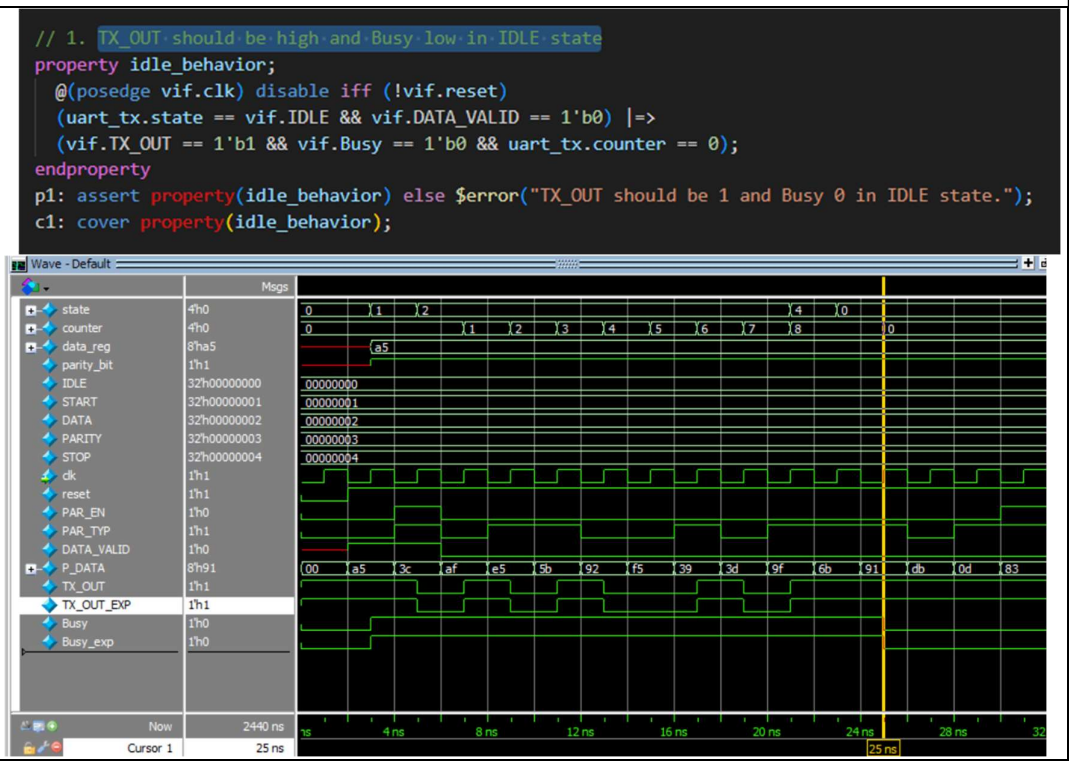
```
UART_TX_IF.sv
UART_TX.sv
UART_TX_GOLDEN.sv
UART_TX_SVA.sv
UART_TX_SEQ_ITEM_pkg.sv
UART_TX_SEQS_pkg.sv
UART_TX_COVERAGE_pkg.sv
UART_TX_DRIVER_pkg.sv
UART_TX_MONITOR_pkg.sv
UART_TX_SCOREBOARD_pkg.sv
UART_TX_SEQUENCER_pkg.sv
UART_TX_AGENT_pkg.sv
UART_TX_GONFIGRATION_pkg.sv
UART_TX_ENVIRONMENT_pkg.sv
UART_TX_TEST_pkg.sv
UART_TX_TOP.sv
```

UVM testbench structure:



2 Assertions table:

Feature	Assertion

When reset is deasserted : State at IDLE ,TX_OUT = 1'B1,Busy = 1'b0,counter=0	
TX_OUT should be high and busy ;ow in IDLE STATE	

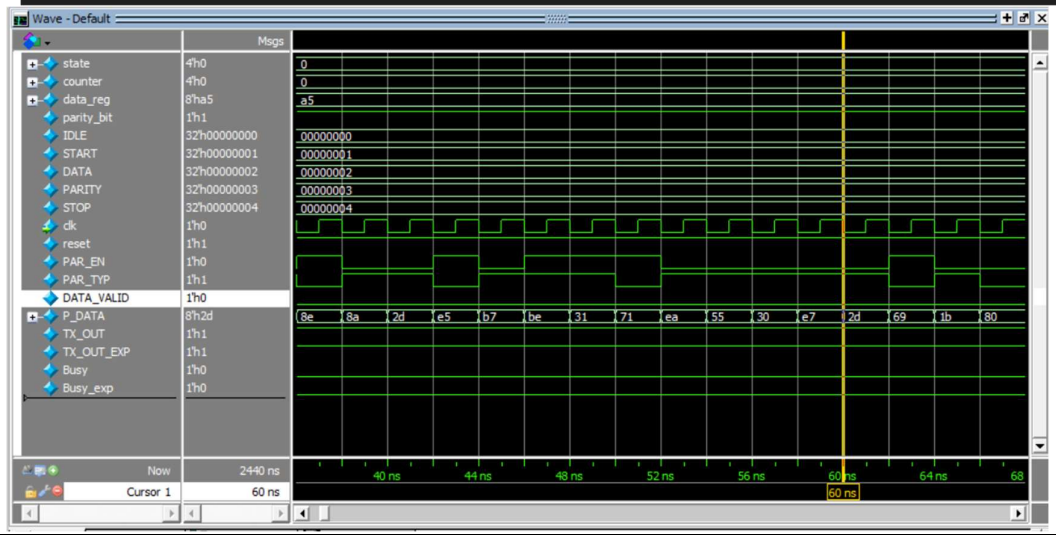
```
// Final reset checks after reset is deasserted
always_comb begin
  if (!vif.reset) begin
    a_assert: assert final(uart_tx.state == vif.IDLE);
    b_assert: assert final(vif.TX_OUT == 1'b1);
    c_assert: assert final(vif.Busy == 1'b0);
    d_assert: assert final(uart_tx.counter == 0);
  end
end
```



```
// 1. TX_OUT should be high and Busy low in IDLE state
property idle_behavior;
  @(posedge vif.clk) disable iff (!vif.reset)
  (uart_tx.state == vif.IDLE && vif.DATA_VALID == 1'b0) |=>
  (vif.TX_OUT == 1'b1 && vif.Busy == 1'b0 && uart_tx.counter == 0);
endproperty
p1: assert property(idle_behavior) else $error("TX_OUT should be 1 and Busy 0 in IDLE state.");
c1: cover property(idle_behavior);
```

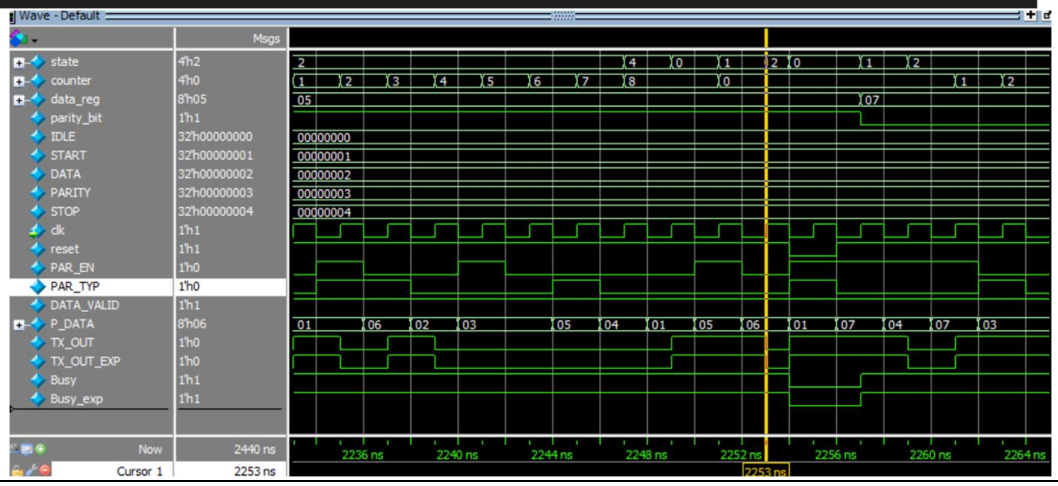
FSM should remain in IDLE when DATA_VALID =0

```
// 2. FSM should remain in IDLE when DATA_VALID = 0
property idle_2;
  @(posedge vif.clk) disable iff (!vif.reset)
  (uart_tx.state == vif.IDLE && vif.DATA_VALID == 1'b0) |=> (uart_tx.state == vif.IDLE);
endproperty
p2: assert property(idle_2) else $error("State should remain IDLE when DATA_VALID is 0.");
c2: cover property(idle_2);
```



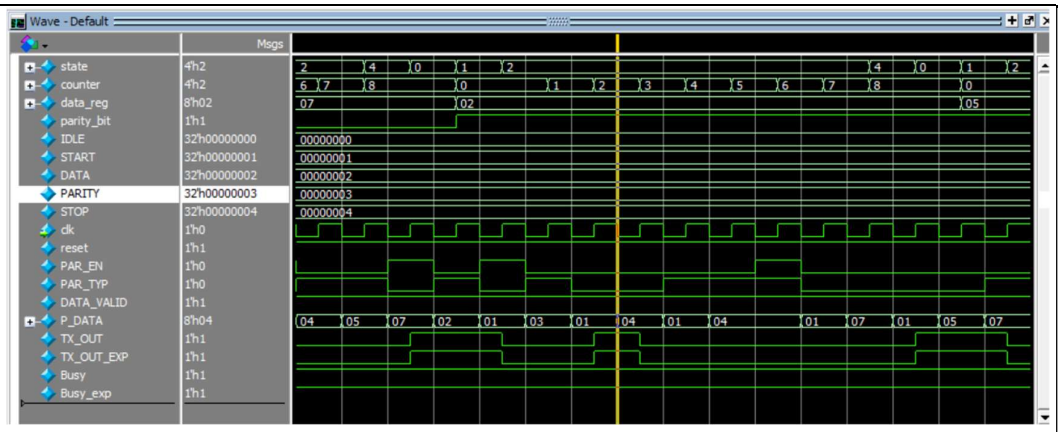
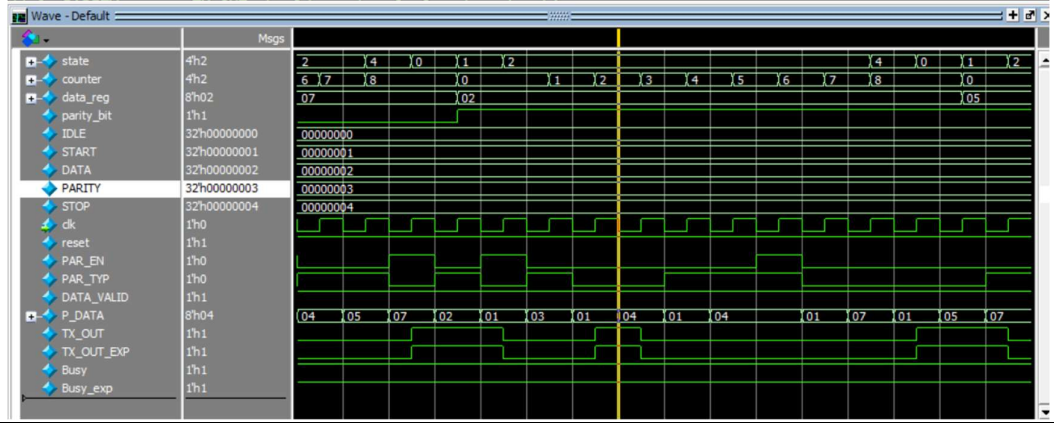
TX_OUT should be 0 and counter =0 at state START

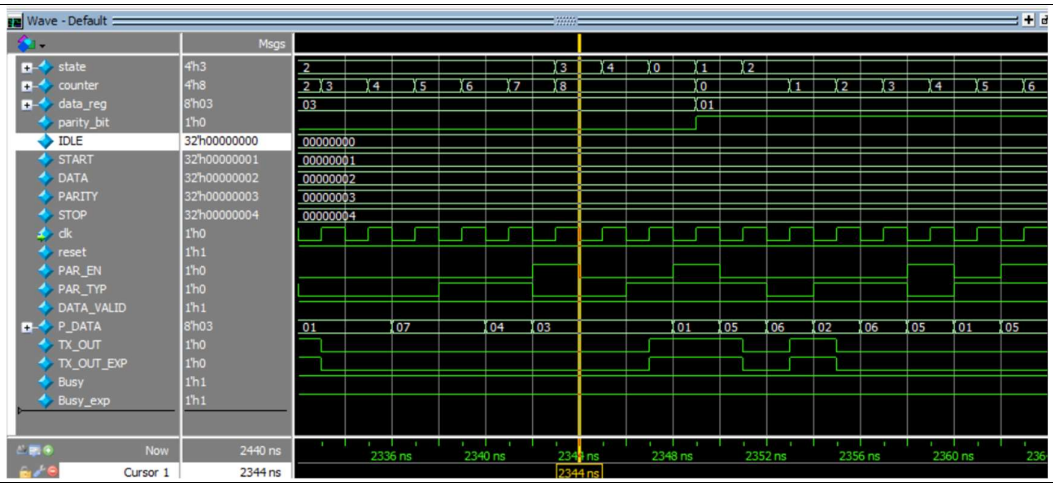
```
// 3. START state: TX_OUT should be 0 and counter = 0
property start_bit_low;
  @(posedge vif.clk) disable iff (!vif.reset)
  (uart_tx.state == vif.START) |=> (vif.TX_OUT == 1'b0 && uart_tx.counter == 0);
endproperty
p3: assert property(start_bit_low) else $error("TX_OUT should be 0 and counter 0 in START state.");
c3: cover property(start_bit_low);
```



DATA state continues when counter not equal 7

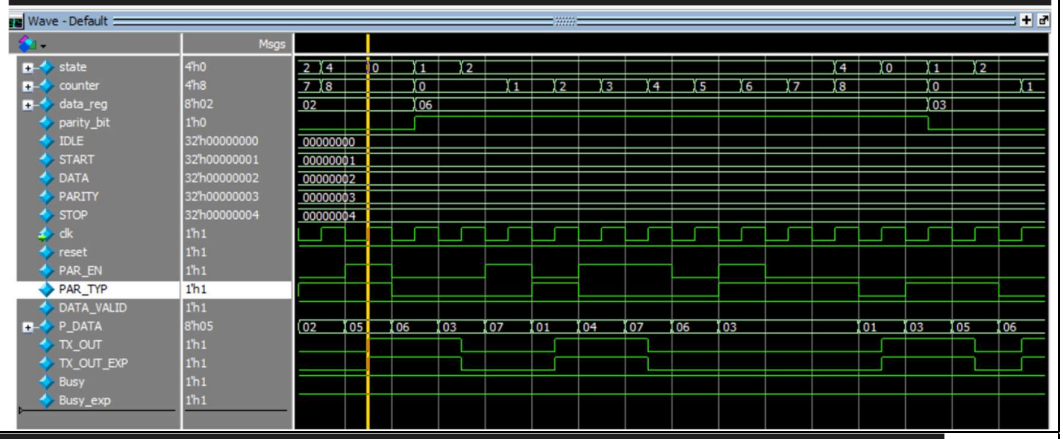
```
// 4. DATA state continues when counter != 7
property data_bit_1;
  @(posedge vif.clk) disable iff (!vif.reset)
  (uart_tx.state == vif.DATA && uart_tx.counter != 7) |=> (uart_tx.state == vif.DATA);
endproperty
p4: assert property(data_bit_1) else $error("FSM should stay in DATA state while counter != 7.");
c4: cover property(data_bit_1);
```

	
TX_OUT should match data_reg[counter] and counter must increment	``` // 5. TX_OUT should match data_reg[counter] and counter must increment property data_bit_match; @(posedge vif.clk) disable iff (!vif.reset) (uart_tx.state == vif.DATA && uart_tx.counter != 7) |=> (vif.TX_OUT == uart_tx.data_reg[$past(uart_tx.counter)] && uart_tx.counter == $past(uart_tx.counter) + 1); endproperty p5: assert property(data_bit_match) else $error("TX_OUT mismatch or counter didn't increment in DATA state."); c5: cover property(data_bit_match); ``` 
PARITY state: TX_OUT should match calculated parity bit	``` // 6. PARITY state: TX_OUT should match calculated parity bit property parity_correct; @(posedge vif.clk) disable iff (!vif.reset) (uart_tx.state == vif.PARITY) |=> (vif.TX_OUT == uart_tx.parity_bit); endproperty p6: assert property(parity_correct) else $error("TX_OUT does not match expected parity bit."); c6: cover property(parity_correct); ```



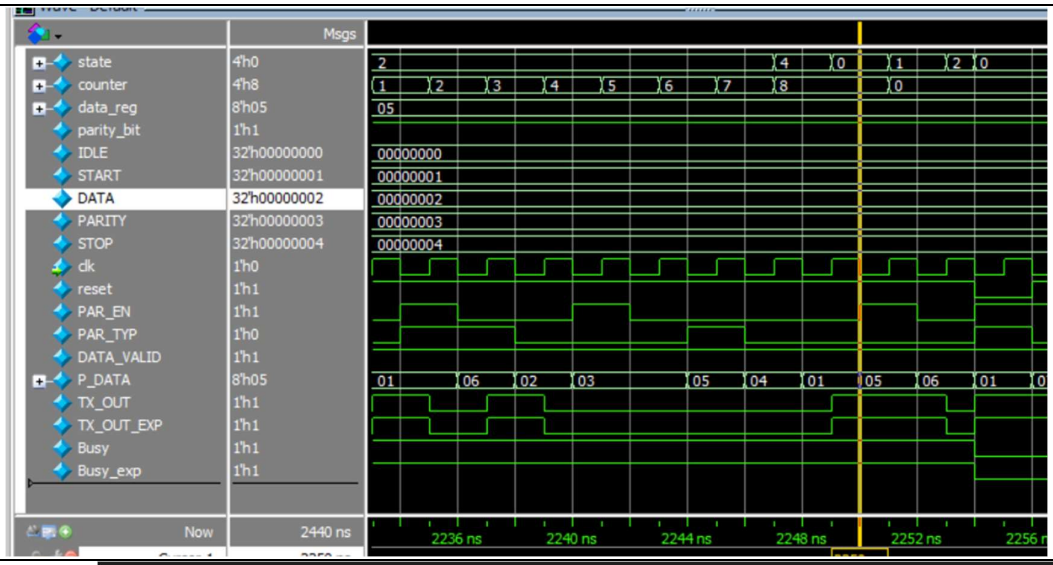
STOP state: TX_OUT should be high.

```
// 7. STOP state: TX_OUT should be high
property stop_bit_high;
  @(posedge vif.clk) disable iff (!vif.reset)
  (uart_tx.state == vif.STOP) |=> (vif.TX_OUT == 1'b1);
endproperty
p7: assert property(stop_bit_high) else $error("TX_OUT must be high in STOP state.");
c7: cover property(stop_bit_high);
```



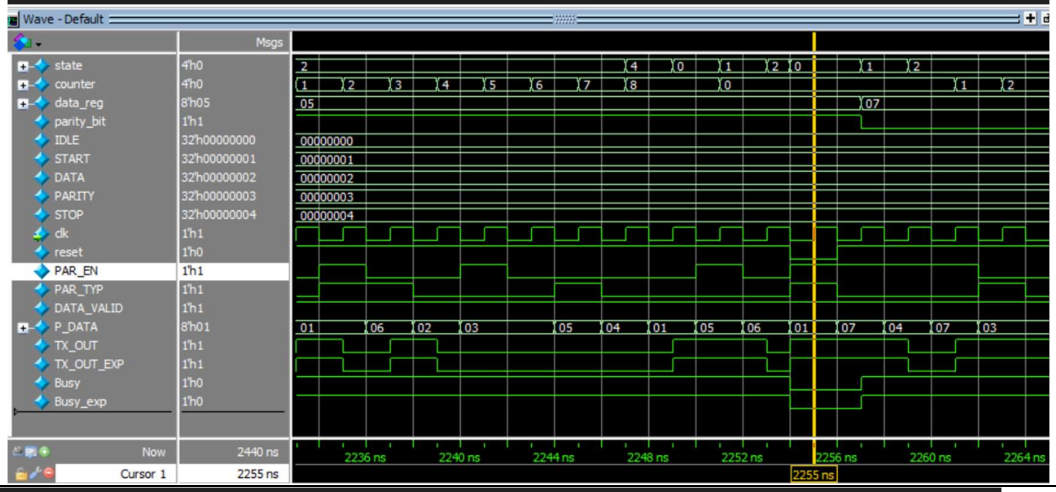
Even parity logic from IDLE.

```
// 9. Even parity logic from IDLE
property idle_even_parity;
  @(posedge vif.clk) disable iff (!vif.reset)
  (uart_tx.state == vif.IDLE && vif.DATA_VALID == 1 && vif.PAR_TYP == 0) |=>
  (uart_tx.parity_bit == ~^$past(vif.P_DATA) &&
   vif.Busy == 1'b1 &&
   uart_tx.data_reg == $past(vif.P_DATA));
endproperty
p9: assert property(idle_even_parity) else $error("Even parity incorrect after IDLE.");
c9: cover property(idle_even_parity);
```



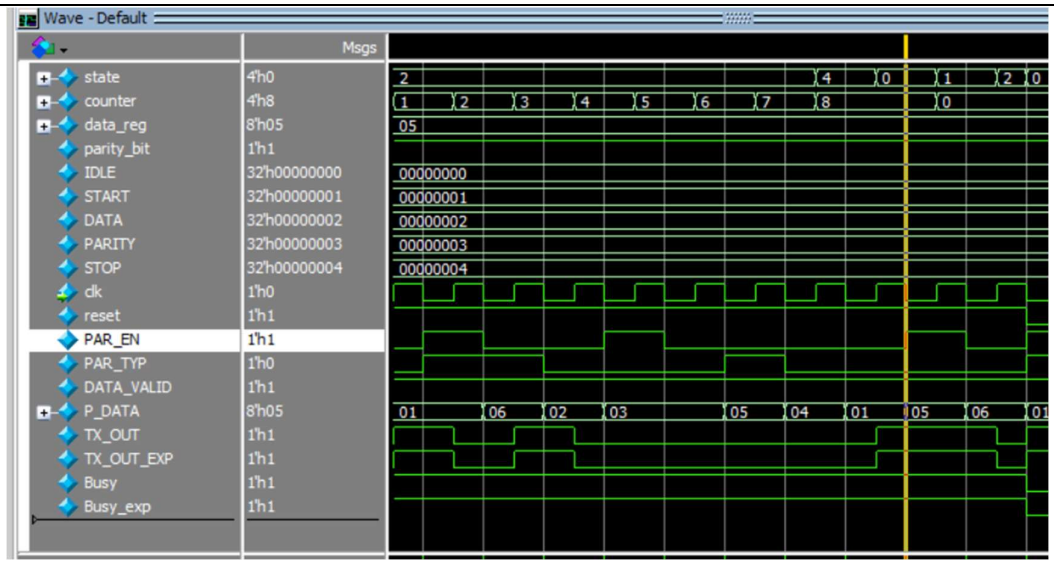
Odd parity logic from IDLE

```
// 10. Odd parity logic from IDLE
property idle_odd_parity;
  @(posedge vif.clk) disable iff (!vif.reset)
  (uart_tx.state == vif.IDLE && vif.DATA_VALID == 1 && vif.PAR_TYP == 1) |=>
  (uart_tx.parity_bit == ^$past(vif.P_DATA) &&
   vif.Busy == 1'b1 &&
   uart_tx.data_reg == $past(vif.P_DATA));
endproperty
p10: assert property(idle_odd_parity) else $error("Odd parity incorrect after IDLE.");
c10: cover property(idle_odd_parity);
```



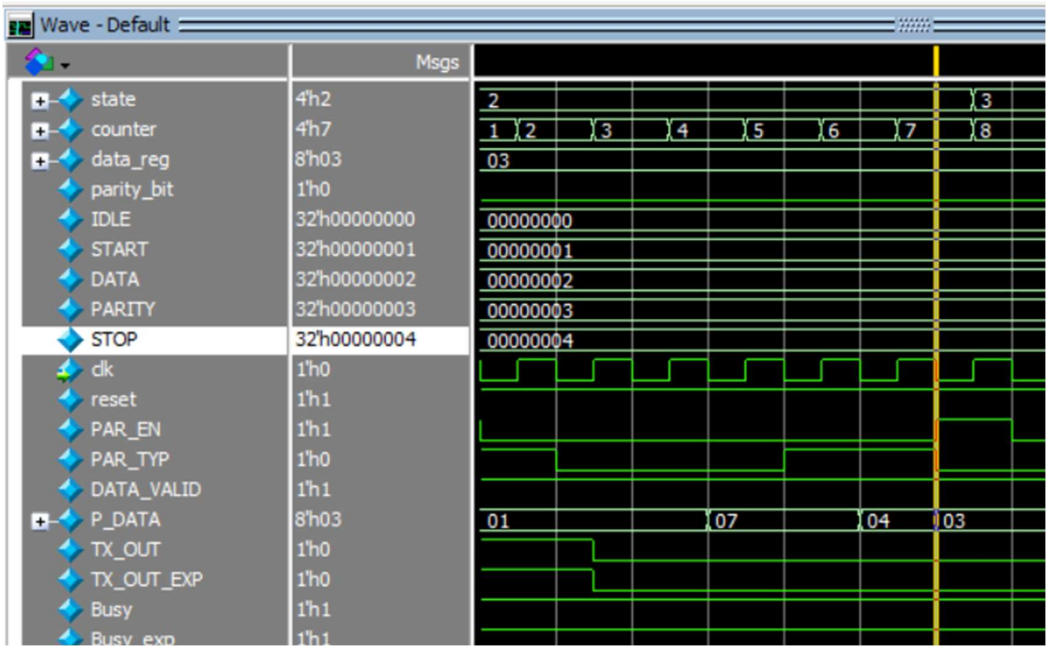
Transition from IDLE to START when DATA_VALID = 1.

```
// 11. Transition from IDLE to START when DATA_VALID = 1
property idle_to_start;
  @(posedge vif.clk) disable iff (!vif.reset)
  (uart_tx.state == vif.IDLE && vif.DATA_VALID == 1) |=> (uart_tx.state == vif.START);
endproperty
p11: assert property(idle_to_start) else $error("Should move from IDLE to START when DATA_VALID = 1.");
c11: cover property(idle_to_start);
```



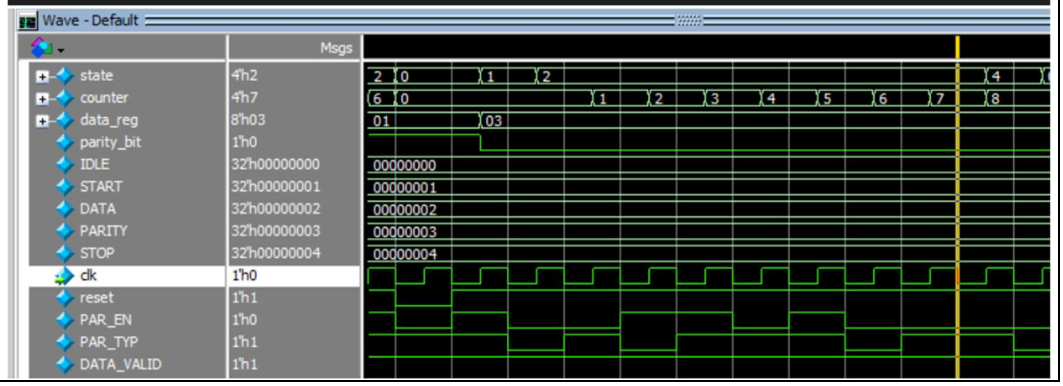
DATA to PARITY transition when counter==7 and PAR_EN==1

```
// 12. DATA to PARITY transition when counter==7 and PAR_EN==1
property data_to_parity;
  @(posedge vif.clk) disable iff (!vif.reset)
  (uart_tx.state == vif.DATA && uart_tx.counter == 7 && vif.PAR_EN == 1) |=>
  (uart_tx.state == vif.PARITY);
endproperty
p12: assert property(data_to_parity) else $error("Should go to PARITY after DATA if PAR_EN == 1.");
c12: cover property(data_to_parity);
```



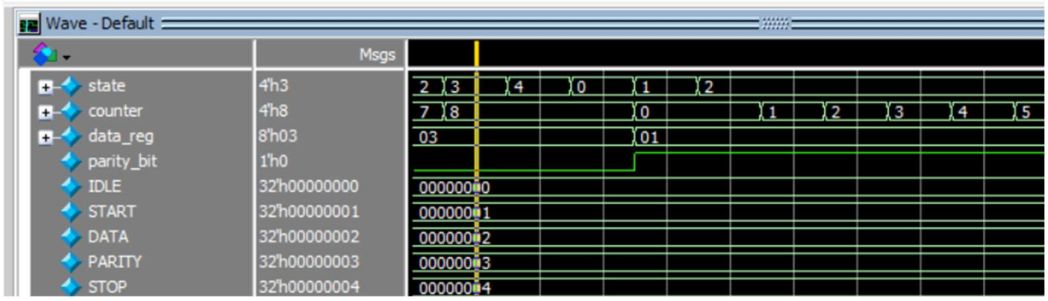
DATA to STOP transition when counter==7 and PAR_EN==0

```
// 13. DATA to STOP transition when counter==7 and PAR_EN==0
property data_to_stop;
  @(posedge vif.clk) disable iff (!vif.reset)
  (uart_tx.state == vif.DATA && uart_tx.counter == 7 && vif.PAR_EN == 0) |=>
  (uart_tx.state == vif.STOP);
endproperty
p13: assert property(data_to_stop) else $error("Should go to STOP after DATA if PAR_EN == 0.");
c13: cover property(data_to_stop);
```



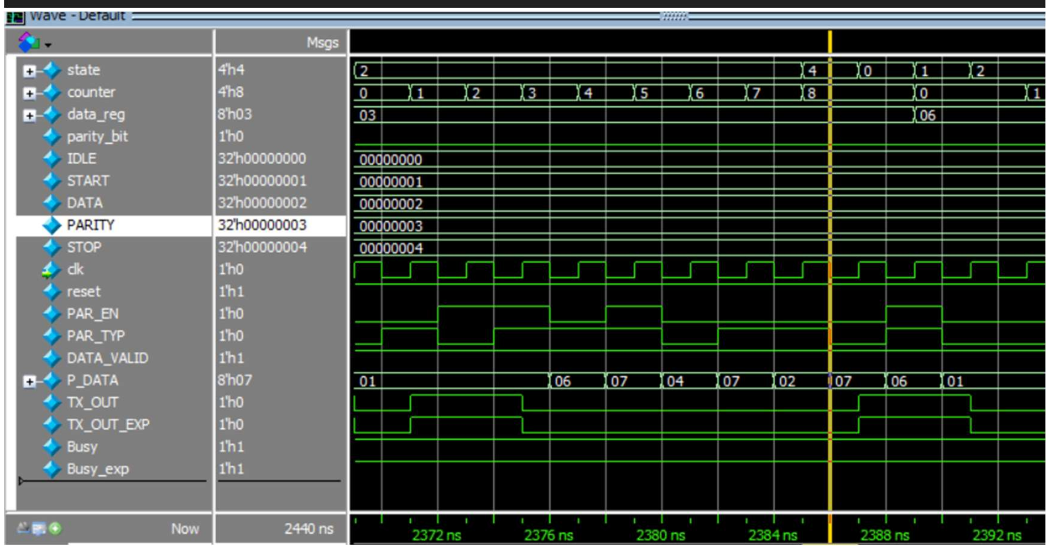
PARITY to STOP transition

```
// 14. PARITY to STOP transition
property parity_to_stop;
  @(posedge vif.clk) disable iff (!vif.reset)
  (uart_tx.state == vif.PARITY) |=>
  (vif.TX_OUT == $past(uart_tx.parity_bit) && uart_tx.state == vif.STOP);
endproperty
p14: assert property(parity_to_stop) else $error("Should go to STOP after PARITY.");
c14: cover property(parity_to_stop);
```



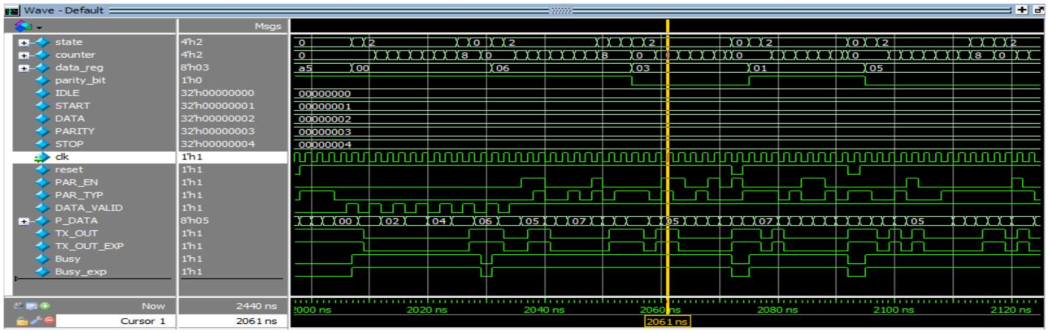
STOP to IDLE transition

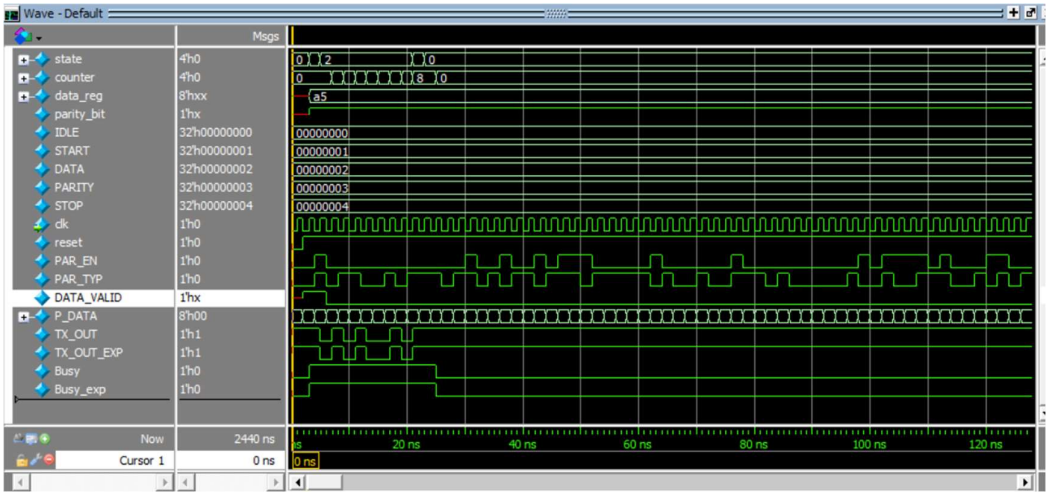
```
// 15. STOP to IDLE transition
property stop_to_idle;
  @(posedge vif.clk) disable iff (!vif.reset)
  (uart_tx.state == vif.STOP) |=>
  (vif.TX_OUT == 1 && uart_tx.state == vif.IDLE);
endproperty
p15: assert property(stop_to_idle) else $error("Should go to IDLE after STOP.");
c15: cover property(stop_to_idle);

endmodule
```

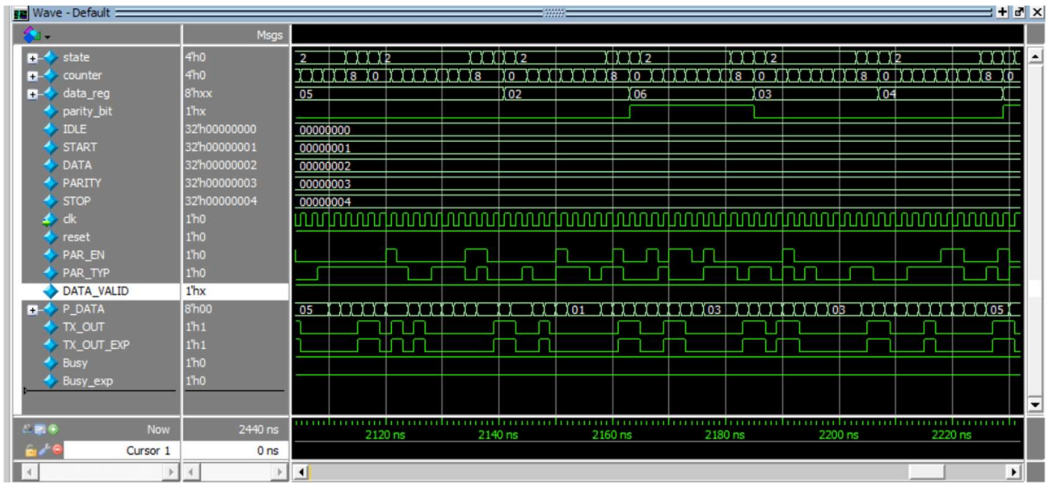


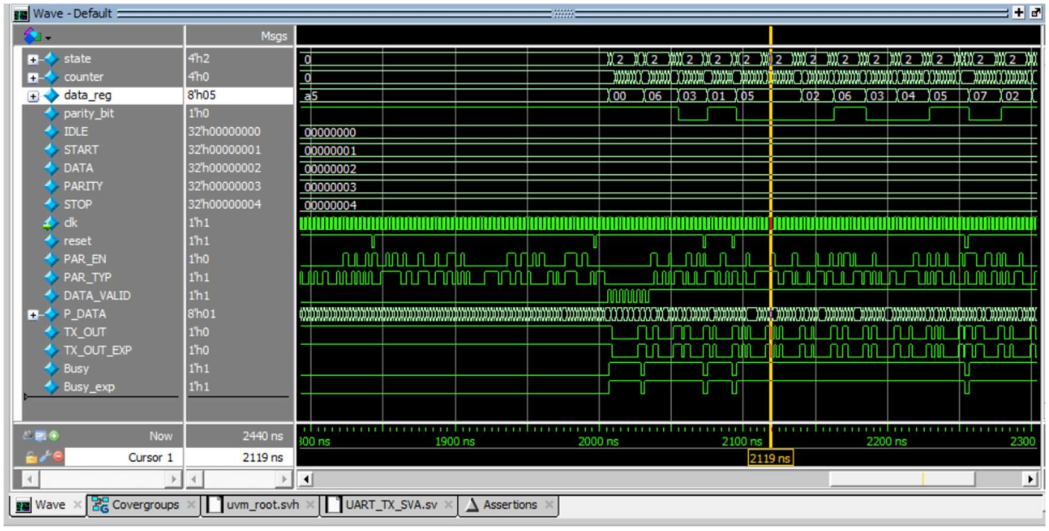
Simulation :

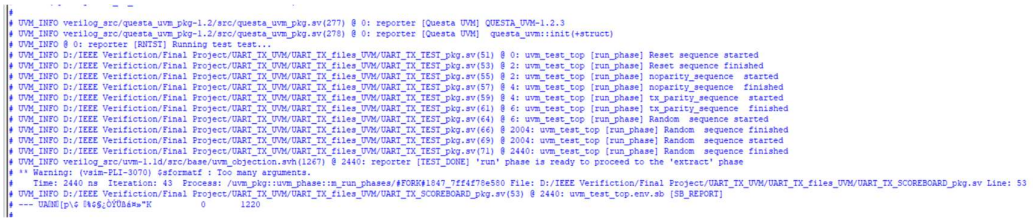
Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Count	Memory	Peak Memory	Peak Memory Time	Cumulative Threads	ATV	Assertion Expression	Included
/uvm_pkg::uvm_re...	Immediate	SVA	on	0	0	-	-	-	-		off	assert ($cast(seq,o))	✗
/uvm_pkg::uvm_re...	Immediate	SVA	on	0	0	-	-	-	-		off	assert ()	✗
/pack_seqs::uart_t...	Immediate	SVA	on	0	1	-	-	-	-		off	assert ()	✓
/pack_seqs::uart_t...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (randomize(...))	✓
/uart_top/DUT/ASS...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (uart_tx.state==0)	✓
/uart_top/DUT/ASS...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (.vif.TX_OUT)	✓
/uart_top/DUT/ASS...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (~vif.Busy)	✓
/uart_top/DUT/ASS...	Immediate	SVA	on	0	1	-	-	-	-		off	assert (uart_tx.counter==0)	✓
/uart_top/DUT/ASS...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge vif.clk) disable i...	✓
/uart_top/DUT/ASS...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge vif.clk) disable i...	✓
/uart_top/DUT/ASS...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge vif.clk) disable i...	✓
/uart_top/DUT/ASS...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge vif.clk) disable i...	✓
/uart_top/DUT/ASS...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge vif.clk) disable i...	✓
/uart_top/DUT/ASS...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge vif.clk) disable i...	✓
/uart_top/DUT/ASS...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge vif.clk) disable i...	✓
/uart_top/DUT/ASS...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge vif.clk) disable i...	✓
/uart_top/DUT/ASS...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge vif.clk) disable i...	✓
/uart_top/DUT/ASS...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge vif.clk) disable i...	✓
/uart_top/DUT/ASS...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge vif.clk) disable i...	✓
/uart_top/DUT/ASS...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge vif.clk) disable i...	✓
/uart_top/DUT/ASS...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge vif.clk) disable i...	✓
/uart_top/DUT/ASS...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge vif.clk) disable i...	✓
/uart_top/DUT/ASS...	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0	off	assert(@(posedge vif.clk) disable i...	✓


Wave - Default
Msgs
state
counter
data_reg
parity_bit
IDLE
START
DATA
PARITY
STOP
clk
reset
PAR_EN
PAR_TYP
DATA_VALID
P_DATA
TX_OUT
TX_OUT_EXP
Busy
Busy_exp
Now
2440 ns
Cursor 1
2061 ns
2020 ns
2040 ns
2060 ns
2080 ns
2100 ns
2120 ns


Wave - Default
Msgs
state
counter
data_reg
parity_bit
IDLE
START
DATA
PARITY
STOP
clk
reset
PAR_EN
PAR_TYP
DATA_VALID
P_DATA
TX_OUT
TX_OUT_EXP
Busy
Busy_exp
Now
2440 ns
Cursor 1
0 ns
20 ns
40 ns
60 ns
80 ns
100 ns
120 ns

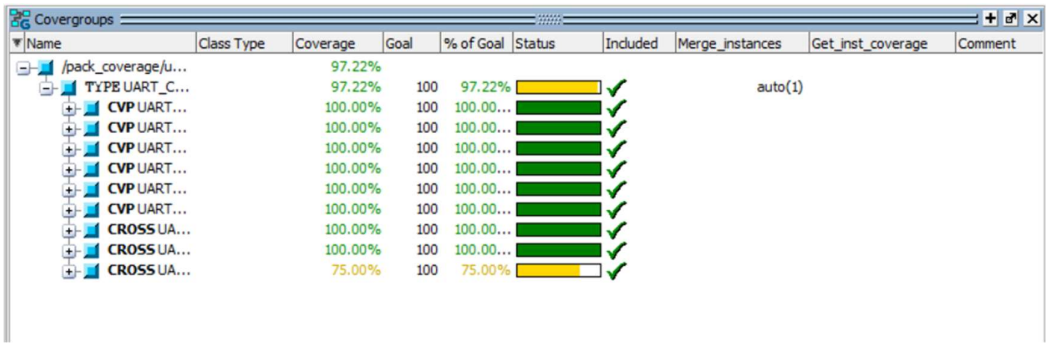

Wave - Default
Msgs
state
counter
data_reg
parity_bit
IDLE
START
DATA
PARITY
STOP
clk
reset
PAR_EN
PAR_TYP
DATA_VALID
P_DATA
TX_OUT
TX_OUT_EXP
Busy
Busy_exp
Now
2440 ns
Cursor 1
0 ns
2120 ns
2140 ns
2160 ns
2180 ns
2200 ns
2220 ns



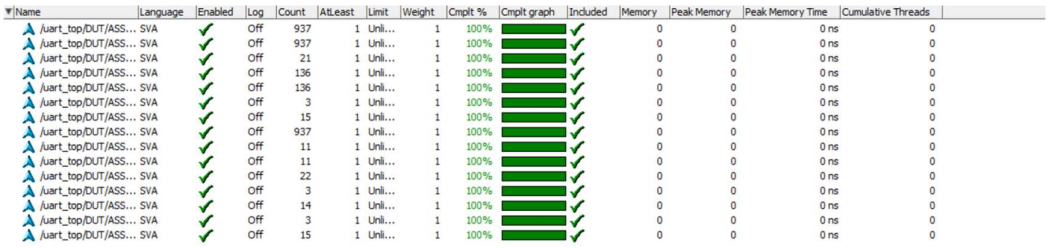


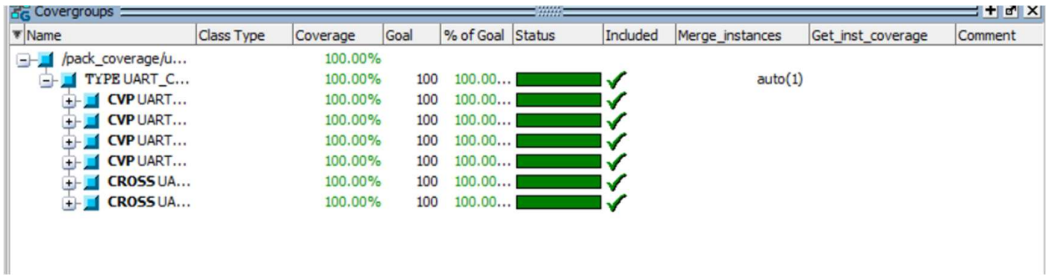
transcript snippets:

Coverage directives:



Code_coverage :

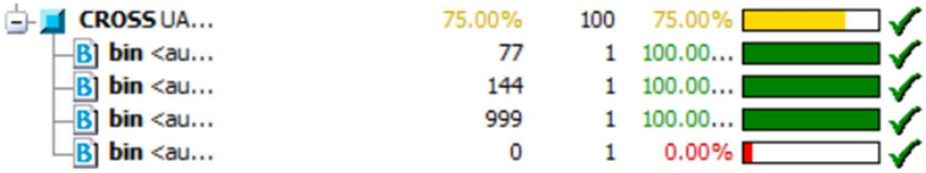




note when we add :

```
cross_TX_OUT_BUSY     : cross seq_item_cov.TX_OUT, seq_item_cov.Busy;
```

All cases of two autputs is not valid :



```
================================================================================
=== Instance: /uart_top/DUT/ASSERTION
=== Design Unit: work.uart_tx_assertions
================================================================================

Directive Coverage:
    Directives                      15        15         0   100.00%

DIRECTIVE COVERAGE:
--------------------------------------------------------------------------------------------
Name                                Design Design   Lang File(Line)      Hits Status
                                    Unit   UnitType
--------------------------------------------------------------------------------------------
/uart_top/DUT/ASSERTION/c1          uart_tx_assertions Verilog  SVA  UART_TX_SVA.sv(20)
                                                                        937 Covered
/uart_top/DUT/ASSERTION/c2          uart_tx_assertions Verilog  SVA  UART_TX_SVA.sv(28)
                                                                        937 Covered
/uart_top/DUT/ASSERTION/c3          uart_tx_assertions Verilog  SVA  UART_TX_SVA.sv(36)
                                                                         21 Covered
/uart_top/DUT/ASSERTION/c4          uart_tx_assertions Verilog  SVA  UART_TX_SVA.sv(44)
                                                                        136 Covered
/uart_top/DUT/ASSERTION/c5          uart_tx_assertions Verilog  SVA  UART_TX_SVA.sv(54)
                                                                        136 Covered
/uart_top/DUT/ASSERTION/c6          uart_tx_assertions Verilog  SVA  UART_TX_SVA.sv(62)
```

```
================================================================================

Covergroup Coverage:
    Covergroups                     1        na        na    97.22%
        Coverpoints/Crosses         9        na        na        na
            Covergroup Bins        41        40         1    97.56%
--------------------------------------------------------------------------------------------------------------
Covergroup                                          Metric       Goal       Bins    Status

--------------------------------------------------------------------------------------------------------------
 TYPE /pack_coverage/uart_coverage/UART_CG          97.22%        100          -    Uncovered
    covered/total bins:                                40         41          -
    missing/total bins:                                 1         41          -
    % Hit:                                         97.56%        100          -
    Coverpoint cp_PAR_EN                          100.00%        100          -    Covered
        covered/total bins:                             2          2          -
        missing/total bins:                             0          2          -
        % Hit:                                    100.00%        100          -
        bin auto[0]                                   947          1          -    Covered
        bin auto[1]                                   273          1          -    Covered
    Coverpoint cp_PAR_TYPE                        100.00%        100          -    Covered
        covered/total bins:                             2          2          -
        missing/total bins:                             0          2          -
        % Hit:                                    100.00%        100          -
        bin auto[0]                                   608          1          -    Covered
        bin auto[1]                                   612          1          -    Covered
    Coverpoint cp_P_VALID                         100.00%        100          -    Covered
        covered/total bins:                             2          2          -
        missing/total bins:                             0          2          -
        % Hit:                                    100.00%        100          -
        bin auto[0]                                  1007          1          -    Covered
        bin auto[1]                                   212          1          -    Covered
    Coverpoint cp_P_DATA                          100.00%        100          -    Covered
        covered/total bins:                             8          8          -
        missing/total bins:                             0          8          -
        % Hit:                                    100.00%        100          -
```

```
TOTAL COVERGROUP COVERAGE: 97.22%  COVERGROUP TYPES: 1

DIRECTIVE COVERAGE:
--------------------------------------------------------------------------------------------
Name                                    Design Design   Lang File(Line)      Hits Status
                                        Unit   UnitType
--------------------------------------------------------------------------------------------
/uart_top/DUT/ASSERTION/c1              uart_tx_assertions Verilog  SVA  UART_TX_SVA.sv(20)
                                                                           937 Covered
/uart_top/DUT/ASSERTION/c2              uart_tx_assertions Verilog  SVA  UART_TX_SVA.sv(28)
                                                                           937 Covered
/uart_top/DUT/ASSERTION/c3              uart_tx_assertions Verilog  SVA  UART_TX_SVA.sv(36)
                                                                            21 Covered
/uart_top/DUT/ASSERTION/c4              uart_tx_assertions Verilog  SVA  UART_TX_SVA.sv(44)
                                                                           136 Covered
/uart_top/DUT/ASSERTION/c5              uart_tx_assertions Verilog  SVA  UART_TX_SVA.sv(54)
                                                                           136 Covered
/uart_top/DUT/ASSERTION/c6              uart_tx_assertions Verilog  SVA  UART_TX_SVA.sv(62)
                                                                             3 Covered
/uart_top/DUT/ASSERTION/c7              uart_tx_assertions Verilog  SVA  UART_TX_SVA.sv(70)
                                                                            15 Covered
/uart_top/DUT/ASSERTION/c8              uart_tx_assertions Verilog  SVA  UART_TX_SVA.sv(78)
                                                                           937 Covered
/uart_top/DUT/ASSERTION/c9              uart_tx_assertions Verilog  SVA  UART_TX_SVA.sv(89)
                                                                            11 Covered
/uart_top/DUT/ASSERTION/c10             uart_tx_assertions Verilog  SVA  UART_TX_SVA.sv(100)
                                                                            11 Covered
/uart_top/DUT/ASSERTION/c11             uart_tx_assertions Verilog  SVA  UART_TX_SVA.sv(108)
                                                                            22 Covered
/uart_top/DUT/ASSERTION/c12             uart_tx_assertions Verilog  SVA  UART_TX_SVA.sv(117)
                                                                             3 Covered
/uart_top/DUT/ASSERTION/c13             uart_tx_assertions Verilog  SVA  UART_TX_SVA.sv(126)
                                                                            14 Covered
/uart_top/DUT/ASSERTION/c14             uart_tx_assertions Verilog  SVA  UART_TX_SVA.sv(135)
                                                                             3 Covered
/uart_top/DUT/ASSERTION/c15             uart_tx_assertions Verilog  SVA  UART_TX_SVA.sv(144)
                                                                            15 Covered

TOTAL DIRECTIVE COVERAGE: 100.00%  COVERS: 15

Total Coverage By Instance (filtered view): 98.61%
```

```
Coverage Report by instance with details

================================================================================
=== Instance: /uart_top/R_if
=== Design Unit: work.uart_tx_if
================================================================================
Toggle Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Toggles                         34        34         0   100.00%

================================Toggle Details================================

Toggle Coverage for instance /uart_top/R_if --

                                          Node      1H->0L      0L->1H  "Coverage"
                                          ----------------------------------------
                                          Busy           1           1      100.00
                                      Busy_exp           1           1      100.00
                                    DATA_VALID           1           1      100.00
                                        PAR_EN           1           1      100.00
                                       PAR_TYP           1           1      100.00
                                    P_DATA[0-7]          1           1      100.00
                                        TX_OUT           1           1      100.00
                                    TX_OUT_EXP           1           1      100.00
                                           clk           1           1      100.00
                                         reset           1           1      100.00

Total Node Count     =         17
Toggled Node Count   =         17
Untoggled Node Count =          0

Toggle Coverage      =     100.00% (34 of 34 bins)
```

```
================================================================================

Assertion Coverage:
    Assertions                      19        19         0   100.00%
--------------------------------------------------------------------------------
Name                 File(Line)                    Failure      Pass
                                                   Count        Count
--------------------------------------------------------------------------------
/uart_top/DUT/ASSERTION/a_assert
                     UART_TX_SVA.sv(6)                   0          1
/uart_top/DUT/ASSERTION/b_assert
                     UART_TX_SVA.sv(7)                   0          1
/uart_top/DUT/ASSERTION/c_assert
                     UART_TX_SVA.sv(8)                   0          1
/uart_top/DUT/ASSERTION/d_assert
                     UART_TX_SVA.sv(9)                   0          1
```

```
UART_TX_SVA.sv(143)                                0          1
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                         2         2         0   100.00%

================================Branch Details================================

Branch Coverage for instance /uart_top/DUT/ASSERTION

    Line         Item                      Count     Source
    ----         ----                      -----     ------
  File UART_TX_SVA.sv
-----------------------------------IF Branch-----------------------------------
    5                                        299     Count coming in to IF
    5              1                          41       if (!vif.reset) begin
                                             258     All False Count
Branch totals: 2 hits of 2 branches = 100.00%



Directive Coverage:
    Directives                      15        15         0   100.00%

DIRECTIVE COVERAGE:
--------------------------------------------------------------------------------------------------
Name                                   Design Design   Lang File(Line)      Hits Status
```

```
                                                                                          15 Covered
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                       1         1         0   100.00%

================================Statement Details================================

Statement Coverage for instance /uart_top/DUT/ASSERTION --

    Line         Item                      Count     Source
    ----         ----                      -----     ------
  File UART_TX_SVA.sv
    1                                                module uart_tx_assertions (uart_tx_if.DUT vi
    2
    3                                                // Final reset checks after reset is deasser
    4              1                           299     always_comb begin


================================================================================
=== Instance: /uart_top/DUT
=== Design Unit: work.uart_tx
================================================================================
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        16        15         1    93.75%

================================Branch Details================================

Branch Coverage for instance /uart_top/DUT

    Line         Item                      Count     Source
    ----         ----                      -----     ------
  File UART_TX.sv
```

```
Expression Coverage:
    Enabled Coverage              Bins    Covered    Misses  Coverage
    ----------------              ----       ----    ------  --------
    Expressions                      3         3         0   100.00%

================================Expression Details================================

Expression Coverage for instance /uart_top/DUT --

  File UART_TX.sv
-----------Focused Expression View (Bimodal)------------
Line       23 Item    1  (~R_if.PAR_TYP? ~^R_if.P_DATA: ^R_if.P_DATA)
Expression totals: 3 of 3 input terms covered = 100.00%

       Input Term   Covered  Reason for no coverage                 Hint
      -----------  --------  -------------------------------------  --------------
     R_if.PAR_TYP         Y
    ~^R_if.P_DATA         Y
     ^R_if.P_DATA         Y

    Rows:   Hits(->0)   Hits(->1)  FEC Target            Non-masking condition(s)

---------  ----------  ----------  --------------------  -------------------------
 Row   1:           0           1  R_if.PAR_TYP_0        -
 Row   2:           1           0  R_if.PAR_TYP_1        -
 Row   3:           1           0  ~^R_if.P_DATA_0       ~R_if.PAR_TYP
 Row   4:           0           1  ~^R_if.P_DATA_1       ~R_if.PAR_TYP
 Row   5:           1           0  ^R_if.P_DATA_0        R_if.PAR_TYP
 Row   6:           0           1  ^R_if.P_DATA_1        R_if.PAR_TYP
```

```
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                      22        22         0   100.00%

================================Statement Details================================

Statement Coverage for instance /uart_top/DUT --

    Line         Item                      Count     Source
    ----         ----                      -----     ------
  File UART_TX.sv
    1                                                module uart_tx (uart_tx_if.DUT R_if);
    2
    3
```

```
Condition Coverage:
    Enabled Coverage              Bins    Covered    Misses  Coverage
    ----------------              ----       ----    ------  --------
    Conditions                       1          1         0   100.00%

================================Condition Details================================

Condition Coverage for instance /uart_top/GO --

  File UART_TX_GOLDEN.sv
----------------Focused Condition View-------------------
Line       37 Item    1  (counter_g == 7)
Condition totals: 1 of 1 input term covered = 100.00%

        Input Term   Covered  Reason for no coverage   Hint
       -----------  --------  ----------------------  --------------
  (counter_g == 7)         Y

     Rows:       Hits  FEC Target            Non-masking condition(s)
 ---------  ---------  --------------------  -------------------------
  Row   1:          1  (counter_g == 7)_0    -
  Row   2:          1  (counter_g == 7)_1    -


Expression Coverage:
    Enabled Coverage              Bins    Covered    Misses  Coverage
    ----------------              ----       ----    ------  --------
    Expressions                      3          3         0   100.00%
```

```
Statement Coverage:
    Enabled Coverage             Bins      Hits    Misses  Coverage
    ----------------             ----      ----    ------  --------
    Statements                     22        22         0   100.00%

================================Statement Details================================

Statement Coverage for instance /uart_top/GO --

    Line         Item                    Count     Source
    ----         ----                    -----     ------
  File UART_TX_GOLDEN.sv
    1                                              module uart_tx_golden (uart_tx_if.GOLDEN R_if);
    2
    3
    4                                                reg [3:0] state_g = R_if.IDLE;
    5                                                reg [3:0] counter_g = 0;
    6                                                reg [7:0] data_reg_g;
    7                                                reg parity_bit_g;
    8
    9              1                    1254        always @(posedge R_if.clk or negedge R_if.reset) begin
    10                                                 if (!R_if.reset) begin
    11             1                      69            state_g <= R_if.IDLE;
    12             1                      69            R_if.TX_OUT_EXP <= 1'b1;
    13             1                      69            R_if.Busy_exp <= 1'b0;
    14             1                      69            counter_g <= 0;
    15                                                 end else begin
    16                                                   case (state_g)
    17                                                     R_if.IDLE: begin
    18             1                     986               R_if.TX_OUT_EXP <= 1'b1;
    19             1                     986               R_if.Busy_exp <= 1'b0;
    20             1                     986             counter_g <=0 ;
    21                                                     if (R_if.DATA_VALID) begin
    22             1                      22                 data_reg_g <= R_if.P_DATA;
    23             1                      22                 parity_bit_g <= (R_if.PAR_TYP == 0 ) ? ~^R_if.P_DATA : ^R_if.P_DATA ;
    24             1                      22                 state_g <= R_if.START;
    25             1                      22                 R_if.Busy_exp <= 1'b1;
    26                                                     end
    27                                                       end
    28                                                     R_if.START: begin
    29             1                      22               R_if.TX_OUT_EXP <= 1'b0;
    30             1                      22               state_g <= R_if.DATA;
    31             1                      22               counter_g <= 0 ;
    32                                                     end
```

تم بحمد الله

﴿ وَأَن لَّيْسَ لِلْإِنسَانِ إِلَّا مَا سَعَىٰ﴾

[النجم: 39]

صدق الله العظيم