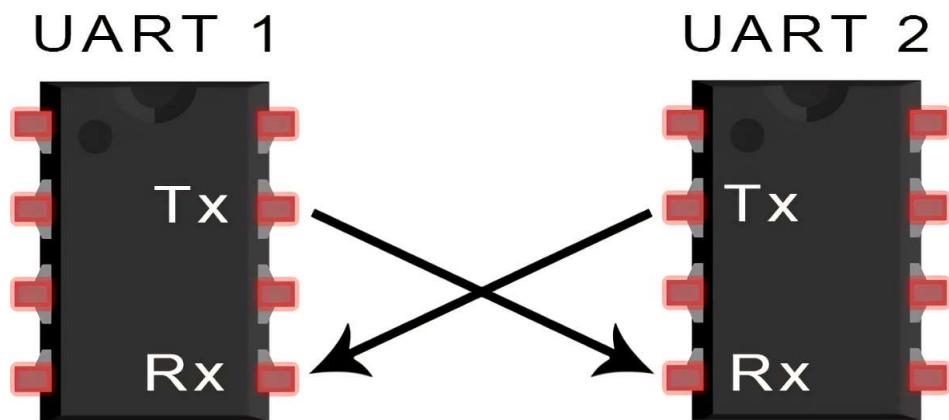


UART_TX

Verify by UVM

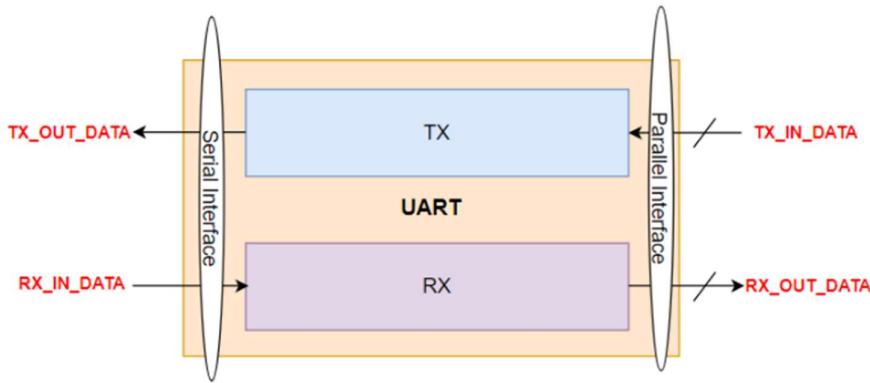
**By: Mohammed Anwar
Abd allatif**



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 Verification > Final Project > UART_TX_UVM > UART_TX_files_UVM >   UART_TX.sv
1  module uart_tx (uart_tx_if.DUT R_if);
2
3
4  reg [3:0] state = R_if.IDLE;
5  reg [3:0] counter = 0;
6  reg [7:0] data_reg;
7  reg parity_bit;
8
9  always @(posedge R_if.clk or negedge R_if.reset) begin
10    if (!R_if.reset) begin
11      state <= R_if.IDLE;
12      R_if.TX_OUT <= 1'b1;
13      R_if.Busy <= 1'b0;
14      counter <= 0;
15    end else begin
16      case (state)
17        R_if.IDLE: begin
18          R_if.TX_OUT <= 1'b1;
19          R_if.Busy <= 1'b0;
20          counter <= 0 ;
21          if (R_if.DATA_VALID) begin
22            data_reg <= R_if.R_DATA;
23            parity_bit <= (R_if.PAR_TYP == 0 ) ? ~^R_if.P_DATA : ^R_if.P_DATA ;
24            state <= R_if.START;
25            R_if.Busy <= 1'b1;
26          end
27        end
28        R_if.START: begin
29          R_if.TX_OUT <= 1'b0;
30          state <= R_if.DATA;
31          counter <= 0 ;
32        end
33      end
34
35      R_if.DATA: begin
36        R_if.TX_OUT <= data_reg[counter];
37        counter <= counter + 1 ;
38        if (counter == 7) begin
39          state <= (R_if.PAR_EN) ? R_if.PARITY : R_if.STOP ;
40        end
41      end
42
43      R_if.PARITY: begin
44        R_if.TX_OUT <= parity_bit ;
45        state <= R_if.STOP;
46      end
47
48      R_if.STOP: begin
49        R_if.TX_OUT <= 1'b1;
50        state <= R_if.IDLE;
51      end
52    endcase
53  end
54 end
55

```

GOLDEN MODEL_CODE :

```

D: > IEEE Verification > Final Project > UART_TX_UVM > UART_TX_files_UVM >  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  always @(posedge R_if.clk or negedge R_if.reset) begin
10    if (!R_if.reset) begin
11      state_g <= R_if.IDLE;
12      R_if.TX_OUT_EXP <= 1'b1;
13      R_if.Busy_exp <= 1'b0;
14      counter_g <= 0;
15    end else begin
16      case (state_g)
17        R_if.IDLE: begin
18          R_if.TX_OUT_EXP <= 1'b1;
19          R_if.Busy_exp <= 1'b0;
20          counter_g <= 0 ;
21          if (R_if.DATA_VALID) begin
22            data_reg_g <= R_if.P_DATA;
23            parity_bit_g <= (R_if.PAR_TYP == 0 ) ? ~^R_if.P_DATA : ^R_if.P_DATA ;
24            state_g <= R_if.START;
25            R_if.Busy_exp <= 1'b1;
26          end
27        end
28        R_if.START: begin
29          R_if.TX_OUT_EXP <= 1'b0;
30          state_g <= R_if.DATA;
31          counter_g <= 0 ;
32        end
33
34        R_if.DATA: begin
35          R_if.TX_OUT_EXP <= data_reg_g[counter_g];
36          counter_g <= counter_g + 1 ;
37          if (counter_g == 7) begin
38            state_g <= (R_if.PAR_EN) ? R_if.PARITY : R_if.STOP ;
39          end
40        end
41
42        R_if.PARITY: begin
43          R_if.TX_OUT_EXP <= parity_bit_g ;
44          state_g <= R_if.STOP;
45        end
46
47        R_if.STOP: begin
48          R_if.TX_OUT_EXP <= 1'b1;
49          state_g <= R_if.IDLE;
50        end
51      endcase
52    end
53  end
54
55 endmodule

```

Interface :

```

D:> IEEE Verification > Final Project > UART_TX_UVM > UART_TX_files_UVM >  UART_TX_IF.sv
1  interface uart_tx_if(clk);
2
3  // parameter Declaration
4  parameter IDLE = 0;
5  parameter START = 1;
6  parameter DATA = 2;
7  parameter PARITY = 3;
8  parameter STOP = 4;
9
10 //----- input declaration -----
11 input bit clk;
12 logic reset ;
13 logic PAR_EN;
14 logic PAR_TYP;
15 logic DATA_VALID;
16 logic [7:0] P_DATA;
17 //----- output declaration -----
18 logic TX_OUT , TX_OUT_EXP;
19 logic Busy , Busy_exp ;
20
21 // _____ MODPORTS _____ //
22
23 modport DUT (input clk , reset , PAR_EN , PAR_TYP , DATA_VALID , P_DATA ,output TX_OUT , Busy );
24
25 modport GOLDEN (input clk , reset , PAR_EN , PAR_TYP , DATA_VALID , P_DATA ,output TX_OUT_EXP , Busy_exp );
26
27
28
29
30
31 endinterface

```

Top module :

```

UART_TX_TOP.sv
1  module UART_TX_TOP ;
2  bit clk;
3  initial begin
4  clk = 0;
5  forever
6  #5 clk = ~clk ;
7
8  end
9
10 uart_tx_if R_if(clk);
11 uart_tx dut(R_if);
12 uart_tx_test tb(R_if);
13 bind uart_tx uart_tx_assertions T1(R_if);
14
15 endmodule

```

UART_TX TEST_pkg:

```

D: > IEEE Verification > Final Project > UART_TX_UVM > UART_TX_files_UVM > |<| UART_TX_TEST_pkg.sv
 1 package pack_test;
 2 import pack_env::*;
 3 import pack_config::*;
 4 import pack_seqs::*;
 5 import uvm_pkg::*;
 6 `include "uvm_macros.svh"
 7
 8 class test extends uvm_test;
 9   `uvm_component_utils(test);
10
11   virtual uart_tx_if uart_vif;
12   uart_config uart_cfg;
13   uart_env env;
14   uart_reset_sequence      seq_one;
15   uart_tx_noparity_sequence seq_two;
16   uart_tx_parity_sequence  seq_three;
17   uart_tx_random_sequence  seq_four;
18   uart_tx_coverage_sequence seq_five;
19
20
21   // Constructor
22   function new(string name="test", uvm_component parent = null);
23     super.new(name, parent);
24   endfunction
25
26   // Build phase
27   function void build_phase(uvm_phase phase);
28     super.build_phase(phase);
29
30     uart_cfg = uart_config::type_id::create("uart_cfg");
31     env = uart_env::type_id::create("env", this);
32     seq_one = uart_reset_sequence::type_id::create("seq_one");
33     seq_two = uart_tx_noparity_sequence::type_id::create("seq_two");
34     seq_three = uart_tx_parity_sequence::type_id::create("seq_three");
35     seq_four = uart_tx_random_sequence::type_id::create("seq_four");
36     seq_five = uart_tx_coverage_sequence::type_id::create("seq_five");
37
38     uvm_config_db#(uart_config)::set(this, "", "CGO", uart_cfg);
39
40     if (!uvm_config_db#(virtual uart_tx_if)::get(this, "", "UART_LL", uart_cfg.uart_vif)) begin
41       `uvm_fatal("build_phase", "Unable to get virtual interface")
42     end
43   endfunction
44
45
46   // Run phase
47   task run_phase(uvm_phase phase);
48     super.run_phase(phase);
49     phase.raise_objection(this);
50
51     `uvm_info("run_phase", "Reset sequence started", UVM_LOW);
52     seq_one.start(env.agt.sqr);
53     `uvm_info("run_phase", "Reset sequence finished", UVM_LOW);
54
55     `uvm_info("run_phase", "noparity_sequence started", UVM_LOW);
56     seq_two.start(env.agt.sqr);
57     `uvm_info("run_phase", "noparity_sequence finished", UVM_LOW);
58
59     `uvm_info("run_phase", "tx_parity_sequence started", UVM_LOW);
60     seq_three.start(env.agt.sqr);
61     `uvm_info("run_phase", "tx_parity_sequence finished", UVM_LOW);
62
63     `uvm_info("run_phase", "Random sequence started", UVM_LOW);
64     seq_four.start(env.agt.sqr);
65     `uvm_info("run_phase", "Random sequence finished", UVM_LOW);
66
67     `uvm_info("run_phase", "Random sequence started", UVM_LOW);
68     seq_five.start(env.agt.sqr);
69     `uvm_info("run_phase", "Random sequence finished", UVM_LOW);
70
71
72     phase.drop_objection(this);
73   endtask
74
75 endclass
76 endpackage

```

UART_TX ENVIRONMENT_pkg:

```

D: > IEEE Verification > 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 Verification > Final Project > UART_TX_UVM > UART_TX_files_UVM >  UART_TX_CONFIGURATION_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 Verification > Final Project > UART_TX_UVM > UART_TX_files_UVM >  UART_TX_SEQ_ITEM_pkg.sv
 1  package pack_seq_item;
 2
 3  import uvm_pkg::*;
 4  `include "uvm_macros.svh"
 5
 6  class shift_reg_seq_item extends uvm_sequence_item;
 7
 8    // FSM state encoding (for future use or ref)
 9    parameter IDLE   = 0;
10    parameter START  = 1;
11    parameter DATA   = 2;
12    parameter PARITY = 3;
13    parameter STOP   = 4;
14
15  `uvm_object_utils(shift_reg_seq_item)
16
17  // Control signals
18  rand logic reset;           // Active-low reset
19  rand logic PAR_EN;          // 0: disable parity, 1: enable parity
20  rand logic PAR_TYP;         // 0: even, 1: odd
21  rand logic DATA_VALID;
22  logic clk;
23
24  // Data
25  rand logic [7:0] P_DATA;
26
27  // DUT outputs
28  logic TX_OUT, TX_OUT_EXP;
29  logic Busy, Busy_exp;
30
31  // Constructor
32  function new(string name = "shift_reg_seq_item");
33  | super.new(name);
34  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=%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 SEQUNCE PKG :

```
D: > IEEE Verification > Final Project > UART_TX_UVM > UART_TX_files_UVM >  UART_TX_SEQUNCE_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); // class 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
14 endpackage
```

UART_TX SEQUENCES PKG :

```
D: > IEEE Verification > 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

```

```

17 //////////////////////////////////////////////////////////////////
18 // Sequence 5: Coverage-oriented sequence
19 //////////////////////////////////////////////////////////////////
20 class uart_tx_coverage_sequence extends uvm_sequence #(shift_reg_seq_item);
21   `uvm_object_utils(uart_tx_coverage_sequence)
22   shift_reg_seq_item seq_item;
23
24   function new(string name = "uart_tx_coverage_sequence");
25     super.new(name);
26   endfunction
27
28   task body();
29     int data_val;
30     int valid;
31     int par_typ;
32
33     // Test all data values from 0 to 7 with DATA_VALID = 0 and 1, PAR_EN = 0
34     for (data_val = 0; data_val <= 7; data_val++) begin
35       for (valid = 0; valid <= 1; valid++) begin
36         seq_item = shift_reg_seq_item::type_id::create("seq_item");
37         start_item(seq_item);
38         seq_item.reset      = 1;
39         seq_item.DATA_VALID = valid;
40         seq_item.PAR_EN     = 0;
41         seq_item.PAR_TYP    = 0;
42         seq_item.P_DATA     = data_val;
43         finish_item(seq_item);
44       end
45     end
46
47     // Check both parity types with PAR_EN = 1
48     for (par_typ = 0; par_typ <= 1; par_typ++) begin
49       seq_item = shift_reg_seq_item::type_id::create("seq_item");
50       start_item(seq_item);
51       seq_item.reset      = 1;
52       seq_item.DATA_VALID = 1;
53       seq_item.PAR_EN     = 1;
54       seq_item.PAR_TYP    = par_typ;
55       seq_item.P_DATA     = 5;
56       finish_item(seq_item);
57     end
58
59     // Generate additional randomized valid stimuli for coverage
60     repeat (200) begin
61       seq_item = shift_reg_seq_item::type_id::create("seq_item");
62       start_item(seq_item);
63       assert(seq_item.randomize()) with {
64         P_DATA inside {[0:7]};
65         DATA_VALID == 1;
66         PAR_EN inside {0,1};
67         PAR_TYP inside {0,1};
68       });
69       finish_item(seq_item);
70     end
71   endtask
72 endclass
73
74 endpackage

```

UART_TX DRIVER_PKG :

```

D:\>IEEE Verification > Final Project > UART_TX_UVM > UART_TX_sequences > #= UART_TX_DRIVER_pkg.sv
 1 package pack_driver;
 2 import pack_config::*;
 3 import uvm_pkg::*;
 4 import pack_seq_item::*;
 5 `include "uvm_macros.svh"
 6 class uart_driver extends uvm_driver #(shift_reg_seq_item);
 7   `uvm_component_utils(uart_driver);
 8   //ram_config ram_cfg;
 9   virtual uart_tx_if sh_vif;
10   shift_reg_seq_item stim_seq_item;
11   //***** COSTRATOR *****/
12   function new(string name ="uart_driver", uvm_component parent = null);
13     super.new(name,parent);
14   endfunction
15   //*****RUN_PHASE***** */
16
17   task run_phase(uvm_phase phase);
18     super.run_phase(phase);
19     forever
20     begin
21       // Create a new sequence item
22       stim_seq_item = shift_reg_seq_item::type_id::create("stim_seq_item");
23
24       // Get transaction from sequencer
25       seq_item_port.get_next_item(stim_seq_item);
26
27       // Drive stimulus to DUT interface
28       sh_vif.reset      = stim_seq_item.reset;
29       sh_vif.PAR_EN     = stim_seq_item.PAR_EN;
30       sh_vif.PAR_TYP    = stim_seq_item.PAR_TYP;
31       sh_vif.P_DATA     = stim_seq_item.P_DATA;
32       sh_vif.DATA_VALID = stim_seq_item.DATA_VALID;
33
34       // Wait for one negative clock edge
35       @(negedge sh_vif.clk);
36
37       // Capture DUT output
38       stim_seq_item.TX_OUT = sh_vif.TX_OUT;
39       stim_seq_item.Busy = sh_vif.Busy;
40
41
42       // Complete transaction
43       seq_item_port.item_done();
44
45       `uvm_info(`run_phase, stim_seq_item.convert2string_stimulus(), UVM_HIGH)
46     end
47   endtask
48
49 endclass
50

```

UART_TX MONITOR_PKG :

```
D:\> IEEE Verification > Final Project > UART_TX_UVM > UART_TX_files_UVM > UART_MONITOR_pkg.sv
 1 package pack_mon;
 2
 3   `include "uvm_macros.svh"
 4   import uvm_pkg::*;
 5   import pack_seq_item::*;
 6
 7   class uart_monitor extends uvm_monitor;
 8     `uvm_component_utils(uart_monitor)
 9
10     virtual uart_tx_if sh_vif;
11     shift_reg_seq_item rsp_seq_item;
12     uvm_analysis_port #(shift_reg_seq_item) mon_ap;
13
14     function new(string name = "uart_monitor", uvm_component parent = null);
15       super.new(name, parent);
16     endfunction
17
18     function void build_phase(uvm_phase phase);
19       super.build_phase(phase);
20       mon_ap = new("mon_ap", this);
21     endfunction
22
23     task run_phase(uvm_phase phase);
24       super.run_phase(phase);
25       forever begin
26         rsp_seq_item = shift_reg_seq_item::type_id::create("rsp_seq_item");
27
28         @(negedge sh_vif.clk);
29
30         rsp_seq_item.reset = sh_vif.reset;
31         rsp_seq_item.PAR_TYP = sh_vif.PAR_TYP;
32         rsp_seq_item.PAR_EN = sh_vif.PAR_EN;
33         rsp_seq_item.DATA_VALID = sh_vif.DATA_VALID;
34         rsp_seq_item.P_DATA = sh_vif.P_DATA;
35         rsp_seq_item.TX_OUT = sh_vif.TX_OUT;
36         rsp_seq_item.Busy = sh_vif.Busy;
37         rsp_seq_item.TX_OUT_EXP = sh_vif.TX_OUT_EXP;
38         rsp_seq_item.Busy_exp = sh_vif.Busy_exp;
39
40         mon_ap.write(rsp_seq_item);
41
42         `uvm_info("run_phase", rsp_seq_item.convert2string(), UVM_HIGH);
43       end
44     endtask
45
46   endclass
47
48 endpackage
49
```

UART_TX AGENT_PKG :

```

D: > IEEE Verification > Final Project > UART_TX_UVM > UART_TX_files_UVM > UART_TX_AGENT_pkg.sv
 1 package pack_agent;
 2   `include "uvm_macros.svh"
 3   import uvm_pkg::*;
 4   import pack_driver::*;
 5   import pack_sequencer::*;
 6   import pack_seq_item::*;
 7   import pack_mon::*;
 8   import pack_config::*;
 9   import pack_seqs::*;

10
11 class uart_agent extends uvm_agent;
12   `uvm_component_utils(uart_agent);

13   uart_config uart_config_obj_driver; // ✓ Correct type
14   uart_driver uart_dr;
15   uart_sequencer sqr;
16   uart_monitor mon;
17   uvm_analysis_port #(shift_reg_seq_item) agt_ap;

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

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

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

26     uart_dr = uart_driver::type_id::create("uart_dr", this);
27     sqr = uart_sequencer::type_id::create("sqr", this);
28     mon = uart_monitor::type_id::create("mon", this);
29     agt_ap = new("agt_ap", this);
30   endfunction

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

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

35     mon.sh_vif = uart_config_obj_driver.uart_vif;
36     mon.mon_ap.connect(agt_ap);
37   endfunction
38 endclass
39
40 endpackage

```

UART_TX COVERAGE_PKG :

```

D: > IEEE Verification > Final Project > UART_TX_UVM > UART_TX_files_UVM > UART_TX_COVERAGE_pkg.sv
 1 package pack_coverage;
 2
 3   import uvm_pkg::*;
 4   `include "uvm_macros.svh"
 5   import pack_seq_item::*;

 6   class uart_coverage extends uvm_component;
 7     `uvm_component_utils(uart_coverage)

 8     uvm_analysis_export #(shift_reg_seq_item) cov_export;
 9     uvm_tlm_analysis_fifo #(shift_reg_seq_item) cov_tlm_o;
10     shift_reg_seq_item seq_item_cov;

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

14     covergroup UART_cg;
15       cp_PAR_EN : coverpoint seq_item_cov.PAR_EN;
16       cp_PAR_TYPE : coverpoint seq_item_cov.PAR_TYP;
17       cp_P_VALID : coverpoint seq_item_cov.DATA_VALID;
18       cp_P_DATA : coverpoint seq_item_cov.P_DATA {
19         bins low_vals[] = {{[0:7]}; // smaller set
20         ignore_bins u = {[8:255]};}
21     }

22     // Cross coverpoints
23     cross_PAR_EN_PAR_TYPE : cross cp_PAR_EN, cp_PAR_TYPE {
24       ignore_bins invalid = binsof(cp_PAR_EN) intersect {0} &&
25       binsof(cp_PAR_TYPE) intersect {1};
26     }

27     cross_P_DATA_P_VALID : cross cp_P_DATA, cp_P_VALID;
28     cross_TX_OUT_BUSY : cross seq_item_cov.TX_OUT, seq_item_cov.Busy;
29   endgroup
30 
```

```
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 Verification > Final Project > UART_TX_UVM > UART_TX_Items_UVM > UART_TX_SCOREBOARD_pkg.sv
 1 package scoreboard;
 2   `include "uvm_macros.svh"
 3   import uvm_pkg::*;
 4   import pack_seq_item::*; // Using RAM-specific sequence item
 5
 6   class uart_scoreboard extends uvm_scoreboard;
 7     `uvm_component_utils(uart_scoreboard)
 8
 9     // Analysis ports and FIFO
10    uvm_analysis_export #(shift_reg_seq_item) sb_export;
11    uvm_tlm_analysis_fifo #(shift_reg_seq_item) sb_fifo;
12    shift_reg_seq_item seq_item_sb;
13
14    // Statistics
15    int error_count = 0;
16    int correct_count = 0;
17    // CONSTRUCTOR
18    function new(string name = "uart_scoreboard", uvm_component parent = null);
19      super.new(name, parent);
20    endfunction
21
22    function void build_phase(uvm_phase phase);
23      super.build_phase(phase);
24      sb_export = new("sb_export", this);
25      sb_fifo = new("sb_fifo", this);
26    endfunction
27
28    function void connect_phase(uvm_phase phase);
29      super.connect_phase(phase);
30      sb_export.connect(sb_fifo.analysis_export);
31    endfunction
32
33
34
35
36
37    task run_phase(uvm_phase phase);
38      super.run_phase(phase);
39      forever
40        begin
41          sb_fifo.get(seq_item_sb);
42          if ((seq_item_sb.TX_OUT != seq_item_sb.TX_OUT_EXP) && (seq_item_sb.Busy != seq_item_sb.Busy_exp)) begin
43            `uvm_error("run_phase",$sformat("compartion failed at reference = %0d and %0d", seq_item_sb.TX_OUT_EXP , seq_item_sb.Busy_exp))
44            error_count++;
45          end
46          else
47            correct_count++;
48        end
49      endtask
50
51    function void report_phase(uvm_phase phase);
52      super.report_phase(phase);
53      `uvm_info("SB_REPORT",
54        $sformat("\n--- UART_TX Scoreboard Report ---\n" +
55        "MisMatches:      %0d\n" +
56        "Match           %0d\n" +
57        "-----", error_count,
58        correct_count),
59      UVM_LOW);
60    endfunction
61  endclass
62 endpackage

```

UART_TX ASSERTIONS:

```

D:\> IEEE Verification > Final Project > UART_TX_UVM > UART_TX_files_UVM > E\UART_TX_SVA.sv
 1  module uart_tx_assertions #(uart_tx_if.BUT vif);
 2
 3  // Final reset checks after reset is deasserted
 4  always_comb begin
 5    if (!vif.reset) begin
 6      a_assert: assert final(uart_tx.state == vif.IDLE);
 7      b_assert: assert final(vif.TX_OUT == 1'b1);
 8      c_assert: assert final(vif.Busy == 1'b0);
 9      d_assert: assert final(uart_tx.counter == 0);
10    end
11  end
12
13  // 1. TX_OUT should be high and Busy low in IDLE state
14  property idle_behavior;
15    @(posedge vif.clk) disable iff (!vif.reset)
16    (uart_tx.state == vif.IDLE && vif.DATA_VALID == 1'b0) |=>
17    (vif.TX_OUT == 1'b1 && vif.Busy == 1'b0 && uart_tx.counter == 0);
18  endproperty
19  p1: assert property(idle_behavior) else $error("TX_OUT should be 1 and Busy 0 in IDLE state.");
20  c1: cover property(idle_behavior);
21
22  // 2. FSM should remain in IDLE when DATA_VALID = 0
23  property idle_2;
24    @(posedge vif.clk) disable iff (!vif.reset)
25    (uart_tx.state == vif.IDLE && vif.DATA_VALID == 1'b0) |=> (uart_tx.state == vif.IDLE);
26  endproperty
27  p2: assert property(idle_2) else $error("State should remain IDLE when DATA_VALID is 0.");
28  c2: cover property(idle_2);
29
30  // 3. START state: TX_OUT should be 0 and counter = 0
31  property start_bit_low;
32    @(posedge vif.clk) disable iff (!vif.reset)
33    (uart_tx.state == vif.START) |=> (vif.TX_OUT == 1'b0 && uart_tx.counter == 0);
34  endproperty
35  p3: assert property(start_bit_low) else $error("TX_OUT should be 0 and counter 0 in START state.");
36  c3: cover property(start_bit_low);
37
38  // 4. DATA state continues when counter != 7
39  property data_bit_1;
40    @(posedge vif.clk) disable iff (!vif.reset)
41    (uart_tx.state == vif.DATA && uart_tx.counter != 7) |=> (uart_tx.state == vif.DATA);
42  endproperty
43  p4: assert property(data_bit_1) else $error("FSM should stay in DATA state while counter != 7.");
44  c4: cover property(data_bit_1);
45
46  // 5. TX_OUT should match data_reg[counter] and counter must increment
47  property data_bit_match;
48    @(posedge vif.clk) disable iff (!vif.reset)
49    (uart_tx.state == vif.DATA && uart_tx.counter != 7) |=>
50    (vif.TX_OUT == uart_tx.data_reg[$past(uart_tx.counter)] &&
51     | uart_tx.counter == $past(uart_tx.counter) + 1);
52  endproperty
53  p5: assert property(data_bit_match) else $error("TX_OUT mismatch or counter didn't increment in DATA state.");
54  c5: cover property(data_bit_match);
55
56  // 6. PARITY state: TX_OUT should match calculated parity bit
57  property parity_correct;
58    @(posedge vif.clk) disable iff (!vif.reset)
59    (uart_tx.state == vif.PARITY) |=> (vif.TX_OUT == uart_tx.parity_bit);
60  endproperty
61  p6: assert property(parity_correct) else $error("TX_OUT does not match expected parity bit.");
62  c6: cover property(parity_correct);
63
64  // 7. STOP state: TX_OUT should be high
65  property stop_bit_high;
66    @(posedge vif.clk) disable iff (!vif.reset)
67    (uart_tx.state == vif.STOP) |=> (vif.TX_OUT == 1'b1);
68  endproperty
69  p7: assert property(stop_bit_high) else $error("TX_OUT must be high in STOP state.");
70  c7: cover property(stop_bit_high);

```

```

1 // 8. No transition from IDLE if DATA_VALID is not asserted
2 property no_start_without_valid;
3   @(posedge vif.clk) disable iff (!vif.reset)
4     (uart_tx.state == vif.IDLE & vif.DATA_VALID == 0) |-> (uart_tx.state == vif.IDLE);
5 endproperty
6 p8: assert property(no_start_without_valid) else $error("FSM left IDLE without DATA_VALID = 1.");
7 c8: cover property(no_start_without_valid);
8
9 // 9. Even parity logic from IDLE
10 property idle_even_parity;
11   @(posedge vif.clk) disable iff (!vif.reset)
12   (uart_tx.state == vif.IDLE & vif.DATA_VALID == 1 & vif.PAR_TYP == 0) |->
13     (uart_tx.parity_bit == ~$past(vif.P_DATA)) &&
14     vif.Busy == 1'b1 &&
15     uart_tx.data_reg == $past(vif.P_DATA));
16 endproperty
17 p9: assert property(idle_even_parity) else $error("Even parity incorrect after IDLE.");
18 c9: cover property(idle_even_parity);
19
20 // 10. Odd parity logic from IDLE
21 property idle_odd_parity;
22   @(posedge vif.clk) disable iff (!vif.reset)
23   (uart_tx.state == vif.IDLE & vif.DATA_VALID == 1 & vif.PAR_TYP == 1) |->
24     (uart_tx.parity_bit == ^$past(vif.P_DATA)) &&
25     vif.Busy == 1'b1 &&
26     uart_tx.data_reg == $past(vif.P_DATA));
27 endproperty
28 p10: assert property(idle_odd_parity) else $error("Odd parity incorrect after IDLE.");
29 c10: cover property(idle_odd_parity);
30
31 // 11. Transition from IDLE to START when DATA_VALID = 1
32 property idle_to_start;
33   @(posedge vif.clk) disable iff (!vif.reset)
34   (uart_tx.state == vif.IDLE & vif.DATA_VALID == 1) |-> (uart_tx.state == vif.START);
35 endproperty
36 p11: assert property(idle_to_start) else $error("Should move from IDLE to START when DATA_VALID = 1.");
37 c11: cover property(idle_to_start);
38
39 // 12. DATA to PARITY transition when counter==7 and PAR_EN==1
40 property data_to_parity;
41   @(posedge vif.clk) disable iff (!vif.reset)
42   (uart_tx.state == vif.DATA & uart_tx.counter == 7 & vif.PAR_EN == 1) |->
43     (uart_tx.state == vif.PARITY);
44 endproperty
45 p12: assert property(data_to_parity) else $error("Should go to PARITY after DATA if PAR_EN == 1.");
46 c12: cover property(data_to_parity);
47
48 // 13. DATA to STOP transition when counter==7 and PAR_EN==0
49 property data_to_stop;
50   @(posedge vif.clk) disable iff (!vif.reset)
51   (uart_tx.state == vif.DATA & uart_tx.counter == 7 & vif.PAR_EN == 0) |->
52     (uart_tx.state == vif.STOP);
53 endproperty
54 p13: assert property(data_to_stop) else $error("Should go to STOP after DATA if PAR_EN == 0.");
55 c13: cover property(data_to_stop);
56
57 // 14. PARITY to STOP transition
58 property parity_to_stop;
59   @(posedge vif.clk) disable iff (!vif.reset)
60   (uart_tx.state == vif.PARITY) |->
61     (vif.TX_OUT == $past(uart_tx.parity_bit) && uart_tx.state == vif.STOP);
62 endproperty
63 p14: assert property(parity_to_stop) else $error("Should go to STOP after PARITY.");
64 c14: cover property(parity_to_stop);
65
66 // 15. STOP to IDLE transition
67 property stop_to_idle;
68   @(posedge vif.clk) disable iff (!vif.reset)
69   (uart_tx.state == vif.STOP) |->
70     (vif.TX_OUT == 1 && uart_tx.state == vif.IDLE);
71 endproperty
72 p15: assert property(stop_to_idle) else $error("Should go to IDLE after STOP.");
73 c15: cover property(stop_to_idle);
74
75 endmodule

```

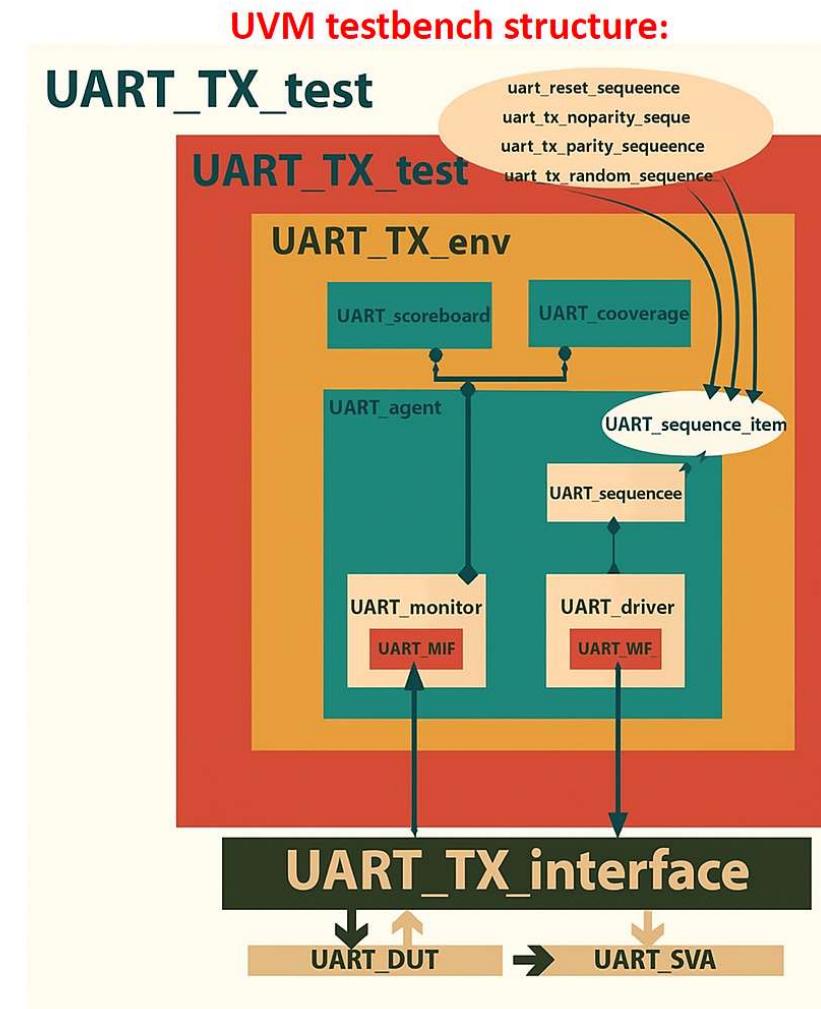
DO_File :

```
D: > IEEE Verification > Final Project > UART_TX_UVM > UART_TX_files_UVM > run.do
1   vlog -sv +cover +acc \
2   UART_TX_IF.sv \
3   UART_TX.sv \
4   UART_TX_GOLDEN.sv \
5   UART_TX_SVA.sv \
6   UART_TX_SEQ_ITEM_pkg.sv \
7   UART_TX_SEQS_pkg.sv \
8   UART_TX_COVERAGE_pkg.sv \
9   UART_TX_DRIVER_pkg.sv \
10  UART_TX_MONITOR_pkg.sv \
11  UART_TX_SCOREBOARD_pkg.sv \
12  UART_TX_SEQUENCER_pkg.sv \
13  UART_TX_AGENT_pkg.sv \
14  UART_TX_GONFIGURATION_pkg.sv \
15  UART_TX_ENVIRONMENT_pkg.sv \
16  UART_TX_TEST_pkg.sv \
17  UART_TX_TOP.sv \
18  +define+UVM_NO_DPI
19
20  vsim -voptargs=+acc work uart_top -classdebug -uvmcontrol=all -coverage
21
22  add wave /uart_top/clk
23  add wave /uart_top/R_if/*
24
25  coverage save uart_tx.ucdb -onexit
26  run -all
27
28  vcover report uart_tx.ucdb -details -all -annotate -output coverage.txt
29
```

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_GONFIGURATION_pkg.sv
UART_TX_ENVIRONMENT_pkg.sv
UART_TX_TEST_pkg.sv
UART_TX_TOP.sv|
```

UVM testbench structure:

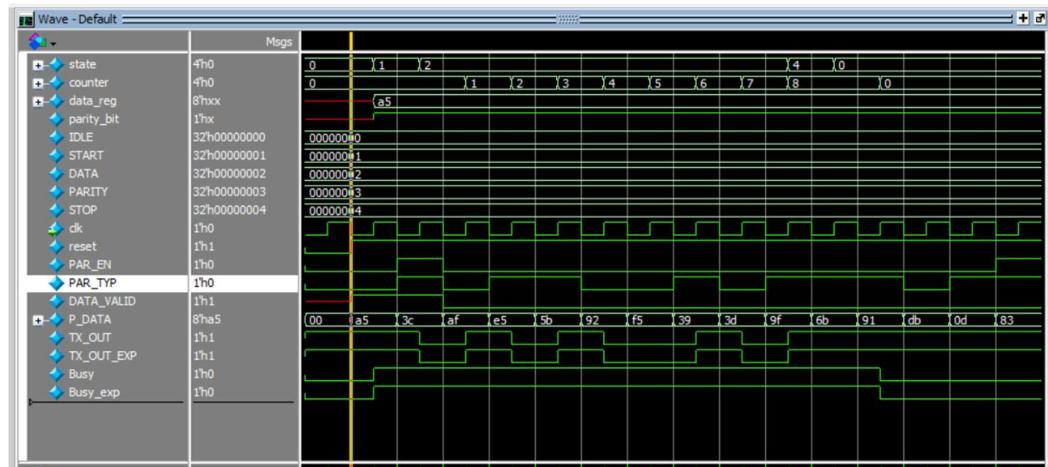


2 Assertions table:

Feature	Assertion
---------	-----------

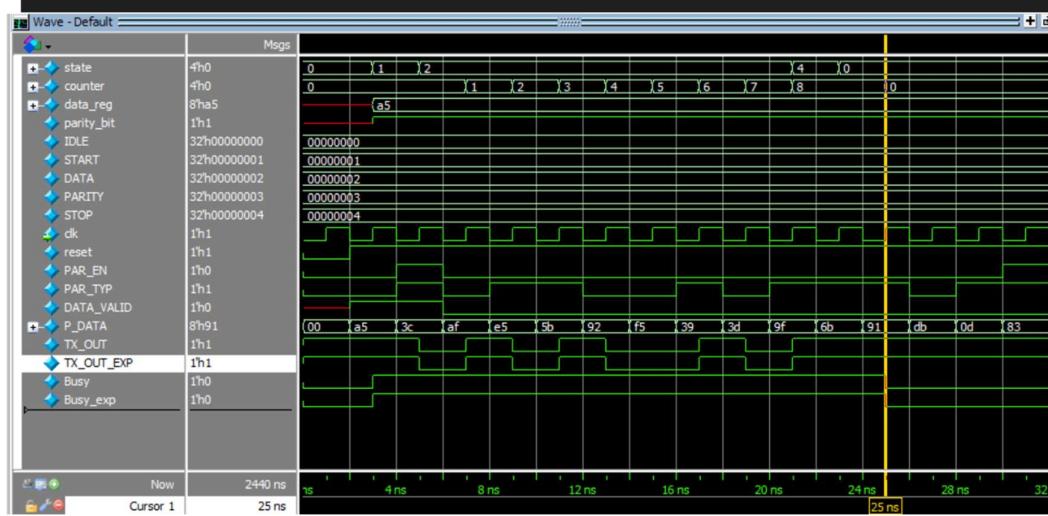
When reset is deasserted :
 State at IDLE
 $\text{TX_OUT} = 1'\text{B}1$, Busy = 1'b0, counter=0

```
// 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
```



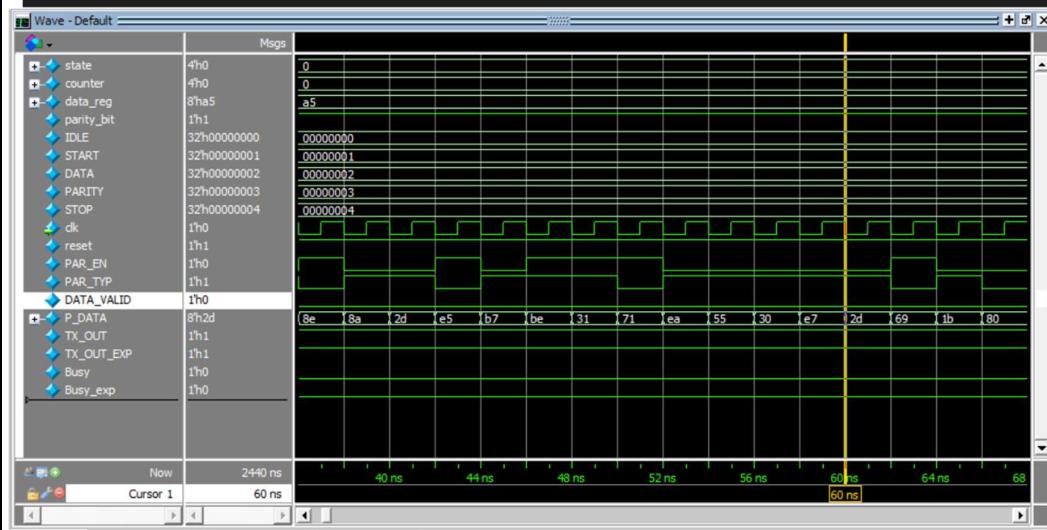
TX_OUT should be high and busy ;ow in IDLE STATE

```
// 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);
```



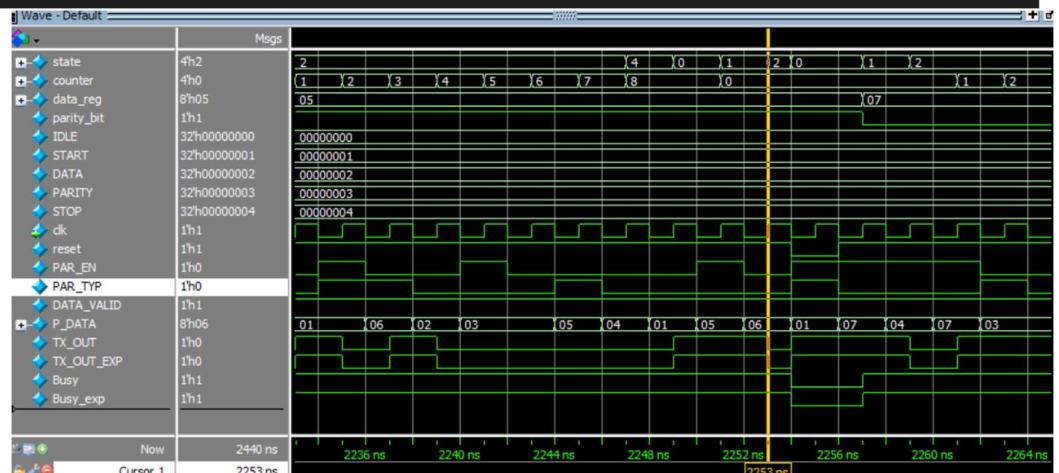
FSM should remain in IDLE when DATA_VALID = 0

```
// 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);
```



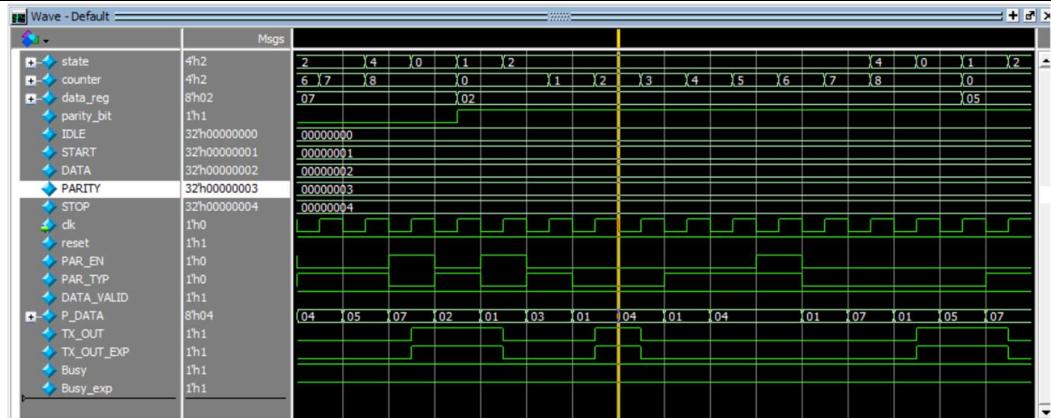
TX_OUT should be 0 and counter = 0 at state START

```
// 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);
```



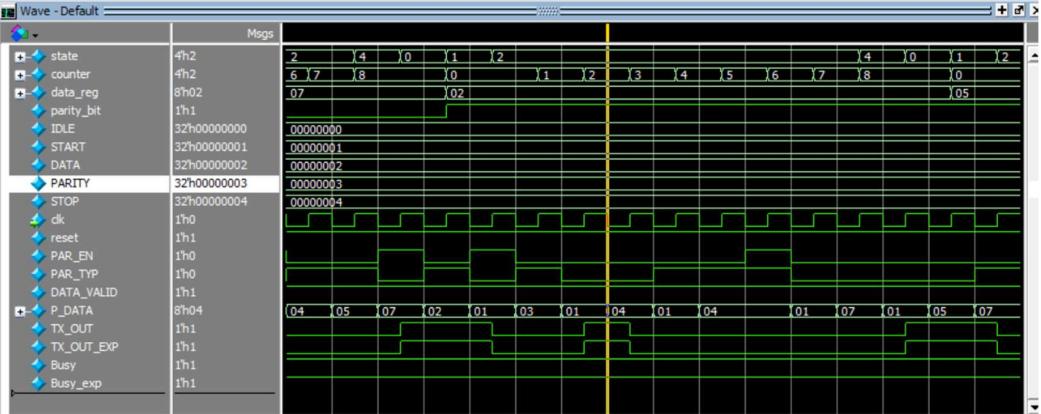
DATA state continues when counter not equal 7

```
// 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);
```



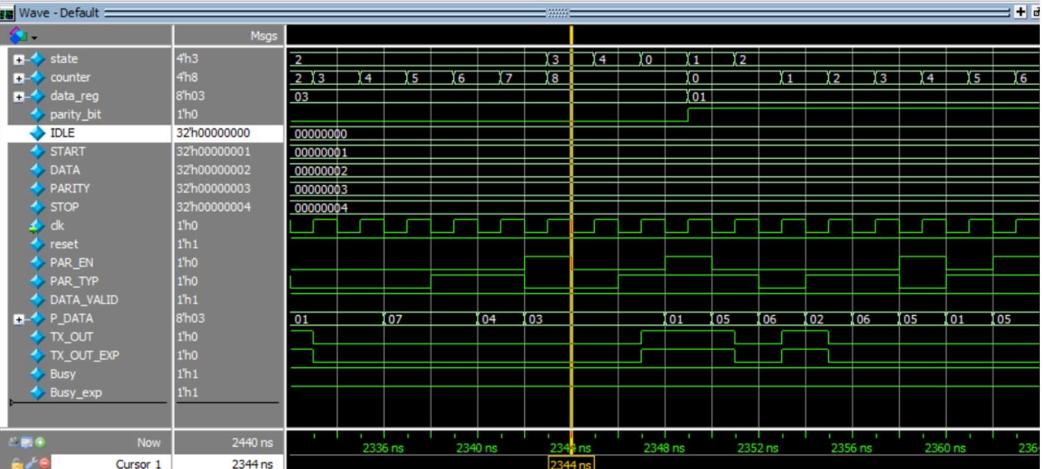
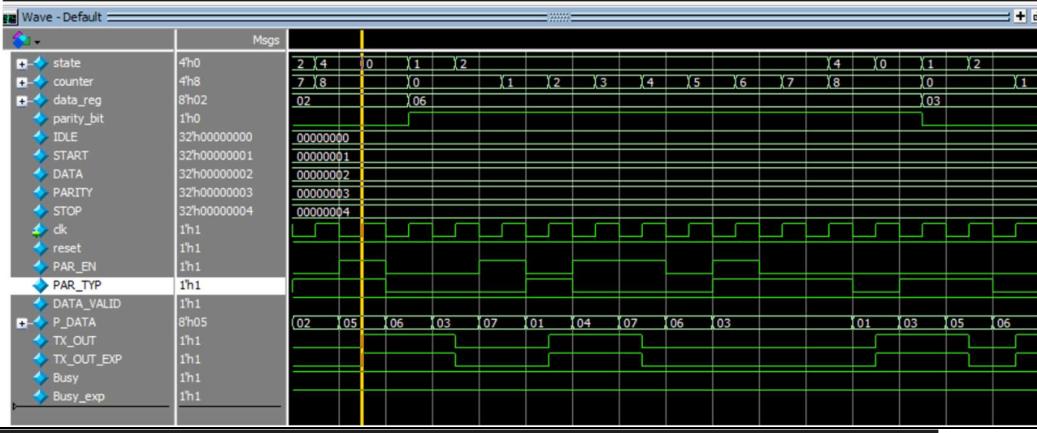
TX_OUT should
match
data_reg[counter]
and counter must
increment

```
// 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);
```

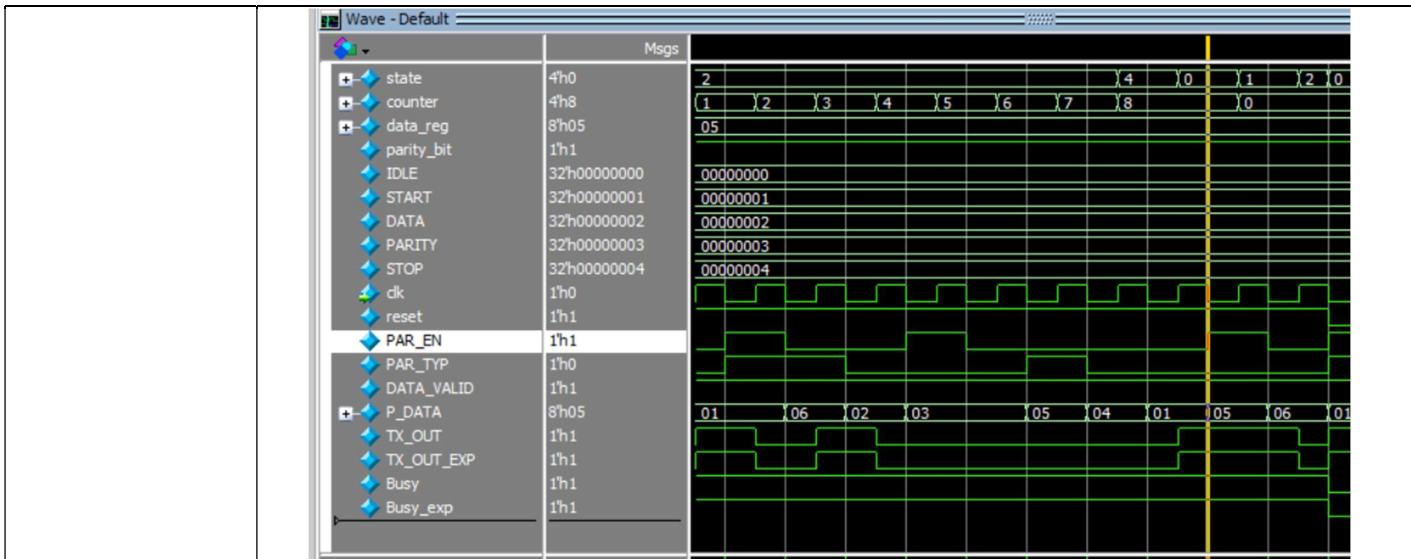


PARITY state: TX_OUT
should match
calculated parity bit

```
// 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);
```

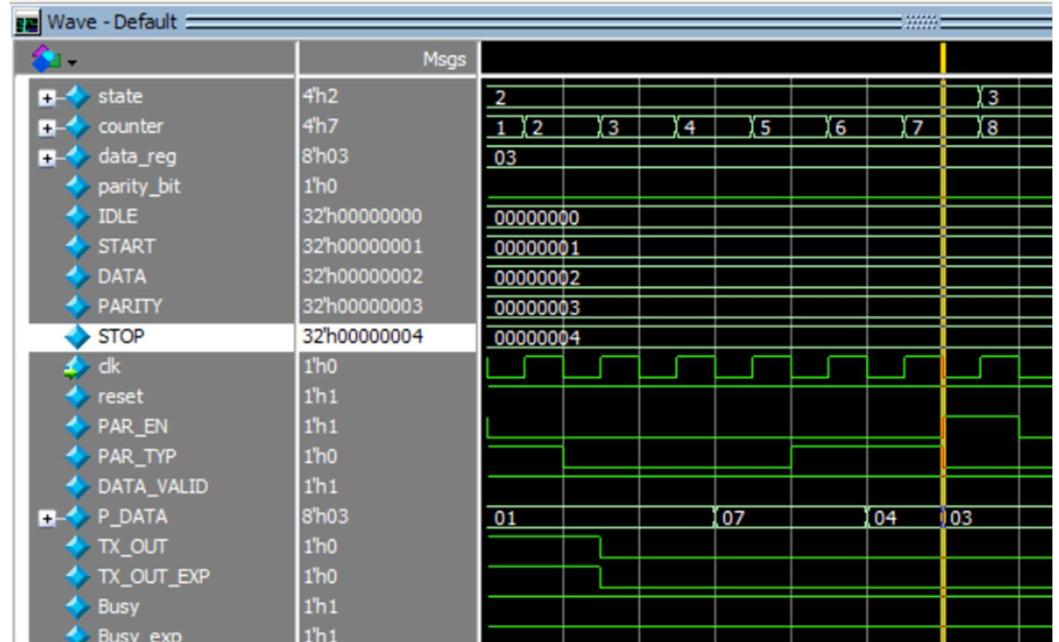
	
STOP state: TX_OUT should be high.	<pre>// 7. STOP state: TX_OUT should be high property stop_bit_high; @(posedge vif.clk) disable iff (!vif.reset) (uart_tx.state == vif.STOP) => (vif.TX_OUT == 1'b1); endproperty p7: assert property(stop_bit_high) else \$error("TX_OUT must be high in STOP state."); c7: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

Odd parity logic from IDLE	<pre>// 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);</pre>
Transition from IDLE to START when DATA_VALID = 1.	<pre>// 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);</pre>



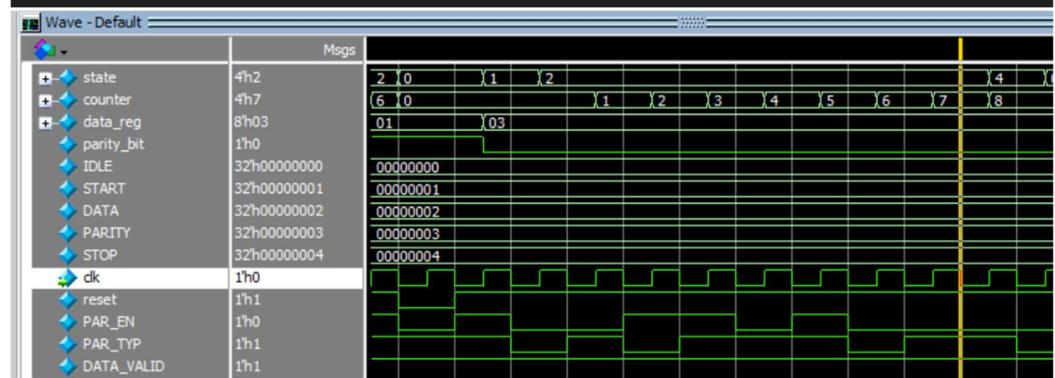
DATA to PARITY
transition when
counter==7 and
PAR_EN==1

```
// 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);
```



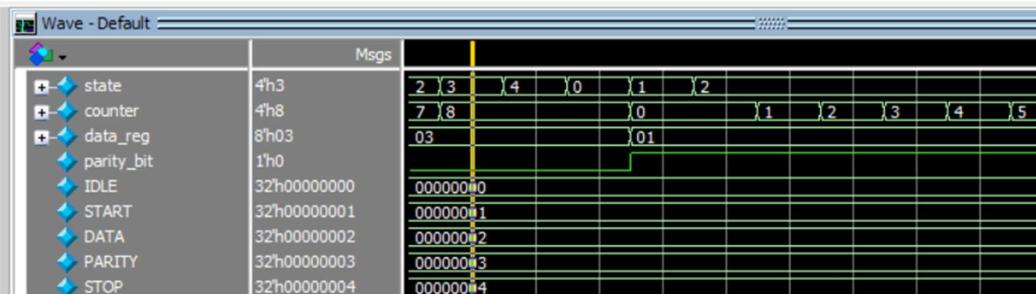
DATA to STOP
transition when
counter==7 and
PAR_EN==0

```
// 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);
```



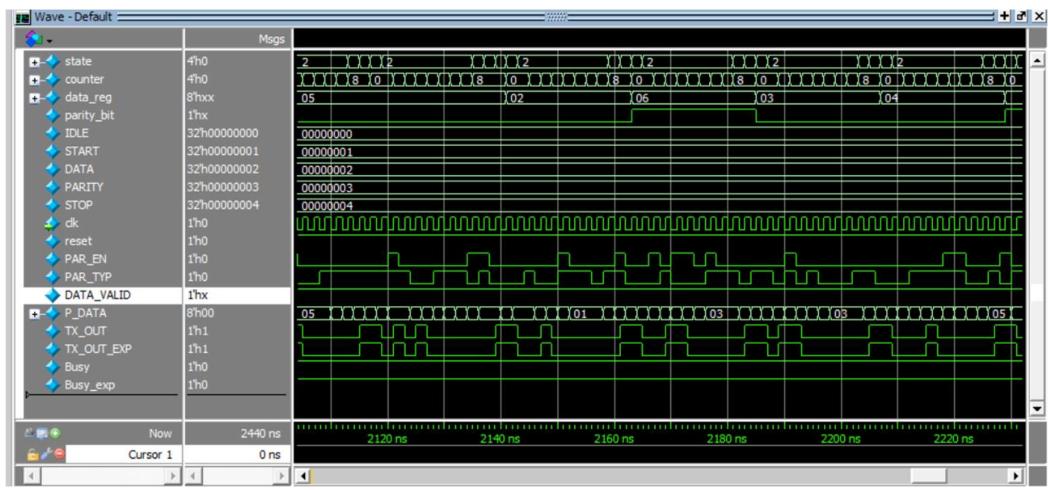
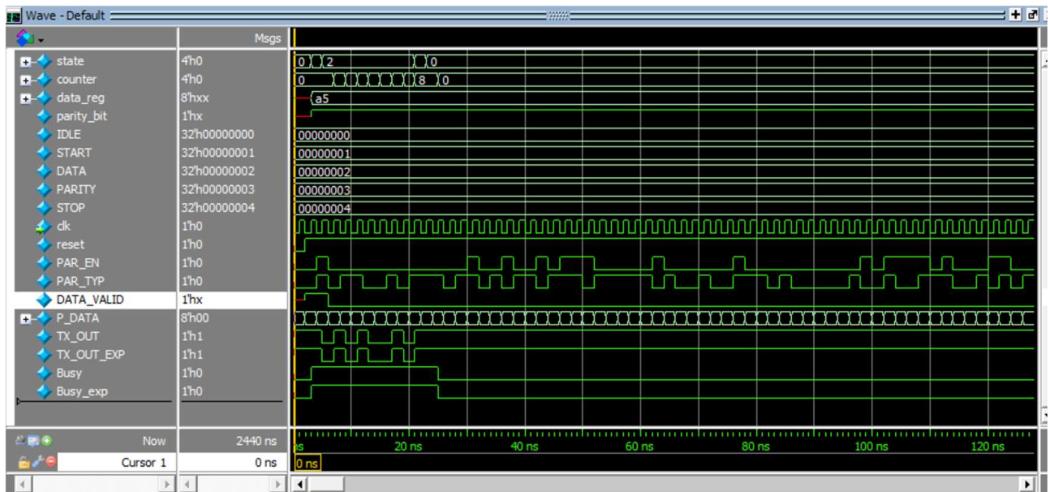
STOP to IDLE transition

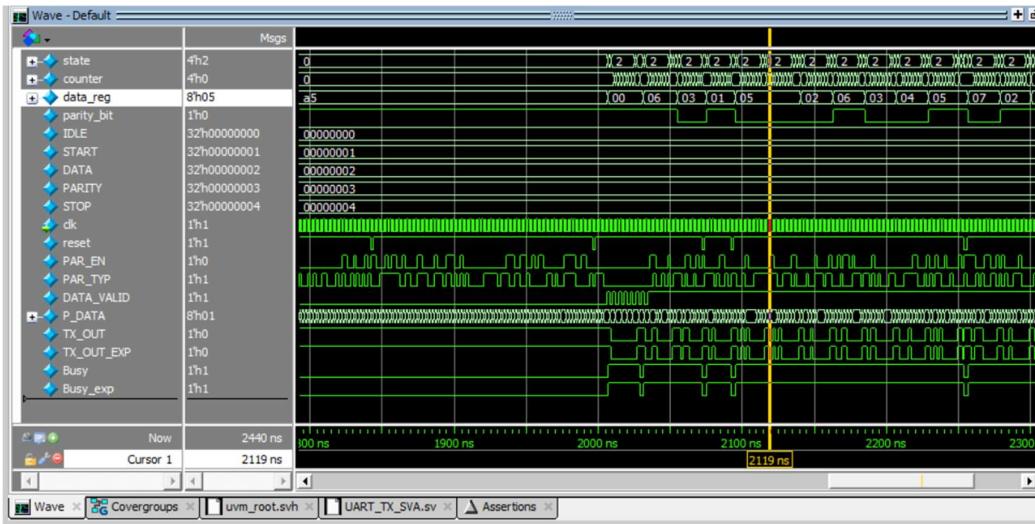
```
// 15. STOP to IDLE transition
property stop_to_idle;
  @(posedge vif.clk) disable iff (!vif.reset)
    (uart_tx.state == vif.STOP) |=>
    (vif.TX_OUT == 1 && uart_tx.state == vif.IDLE);
endproperty
p15: assert property(stop_to_idle) else $error("Should go to IDLE after STOP.");
c15: cover property(stop_to_idle);

endmodule
```

The screenshot shows a waveform simulation in ModelSim. The left pane lists the signals being monitored: 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, and Busy_exp. The right pane displays the waveforms over time. The state signal changes from 0x04 to 0x00. The TX_OUT signal is high during the transition. The clk and reset signals are stable. The P_DATA signal has a value of 0x01. The TX_OUT_EXP signal also changes during the transition.

Simulation :





```

# UVM_INFO verilog_src/questa_umb_pkg.v(277) @ 0: reporter [Questa UVM] QUESTA_UVM-1.2.3
# UVM_INFO verilog_src/questa_umb_pkg.v(1-2/src/questa_umb_pkg.sv(278)) @ 0: reporter [Questa UVM] questauvm::init()
# UVM_INFO @ 0: reporter [RNTIST] Running test test...
# UVM_INFO D:/IEEE Verification/Final Project/UART_TX_Vm/UART_TX_files/UVM/UART_TX_TEST.pkg.sv(11) @ 0: uvm_test_top [run_phase] Reset sequence started
# UVM_INFO D:/IEEE Verification/Final Project/UART_TX_Vm/UART_TX_files/UVM/UART_TX_TEST.pkg.sv(53) @ 2: uvm_test_top [run_phase] Reset sequence finished
# UVM_INFO D:/IEEE Verification/Final Project/UART_TX_Vm/UART_TX_files/UVM/UART_TX_TEST.pkg.sv(55) @ 2: uvm_test_top [run_phase] noparity_sequence started
# UVM_INFO D:/IEEE Verification/Final Project/UART_TX_Vm/UART_TX_files/UVM/UART_TX_TEST.pkg.sv(57) @ 4: uvm_test_top [run_phase] noparity_sequence finished
# UVM_INFO D:/IEEE Verification/Final Project/UART_TX_Vm/UART_TX_files/UVM/UART_TX_TEST.pkg.sv(59) @ 4: uvm_test_top [run_phase] tx_parity_sequence started
# UVM_INFO D:/IEEE Verification/Final Project/UART_TX_Vm/UART_TX_files/UVM/UART_TX_TEST.pkg.sv(61) @ 6: uvm_test_top [run_phase] tx_parity_sequence finished
# UVM_INFO D:/IEEE Verification/Final Project/UART_TX_Vm/UART_TX_files/UVM/UART_TX_TEST.pkg.sv(64) @ 6: uvm_test_top [run_phase] Random sequence started
# UVM_INFO D:/IEEE Verification/Final Project/UART_TX_Vm/UART_TX_files/UVM/UART_TX_TEST.pkg.sv(66) @ 2000: uvm_test_top [run_phase] Random sequence finished
# UVM_INFO D:/IEEE Verification/Final Project/UART_TX_Vm/UART_TX_files/UVM/UART_TX_TEST.pkg.sv(68) @ 2000: uvm_test_top [run_phase] Random sequence started
# UVM_INFO D:/IEEE Verification/Final Project/UART_TX_Vm/UART_TX_files/UVM/UART_TX_TEST.pkg.sv(70) @ 2440: uvm_test_top [run_phase] Random sequence finished
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objectection.svh(1247) @ 2440: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
** Warning: (vsim-PLI-3070) $formatf: Too many arguments.
Time: 2440 ns Iteration: 43 Process: /uvm_pkg/uvm_phase::m_run_phases::#FORW#1847_7ff4f78e580 File: D:/IEEE Verification/Final Project/UART_TX_UVM/UART_TX_files_UVM/UART_TX_SCOREBOARD_pkg.sv Line: 53
UVM_INFO D:/IEEE Verification/Final Project/UART_TX_Vm/UART_TX_files_UVM/UART_TX_SCOREBOARD_pkg.sv(53) @ 2440: uvm_test_top.env.s0 ($0_REPORT)
--- UVM(p) 1456,07Mar2024 1240

```

transcript snippets:

```

# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO : 15
# UVM_WARNING : 0
# UVM_ERROR : 0
# UVM_FATAL : 0
# ** Report counts by id
# [Questa UVM] 2
# [RNTIST] 1
# [SB_REPORT] 1
# [TEST_DONE] 1
# [run_phase] 10
# ** Note: $finish : C:/questasim64_2024.1/win64/..//verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#   Time: 2440 ns Iteration: 61 Instance: /uart_top
# 1
# Break in Task uvm_pkg/uvm_root::run_test at C:/questasim64_2024.1/win64/..//verilog_src/uvm-1.1d/src/base/uvm_root.svh line 430
# Causality operation skipped due to absence of debug database file
VSIM 5>

```

Coverage directives:

Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment
/pack_coverage/u...		97.22%							
- TYPE UART_C...		97.22%	100	97.22%		✓			auto(1)
- CVP UART...		100.00%	100	100.00...		✓			
- CVP UART...		100.00%	100	100.00...		✓			
- CVP UART...		100.00%	100	100.00...		✓			
- CVP UART...		100.00%	100	100.00...		✓			
- CVP UART...		100.00%	100	100.00...		✓			
- CVP UART...		100.00%	100	100.00...		✓			
- CROSS UA...		100.00%	100	100.00...		✓			
- CROSS UA...		100.00%	100	100.00...		✓			
- CROSS UA...		75.00%	100	75.00%		✓			

Code_coverage :

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
uart_top/DUT/ASS... SVA		✓	Off	937	1	Unl...	1	100%		✓	0	0	0 ns	0
uart_top/DUT/ASS... SVA		✓	Off	937	1	Unl...	1	100%		✓	0	0	0 ns	0
uart_top/DUT/ASS... SVA		✓	Off	21	1	Unl...	1	100%		✓	0	0	0 ns	0
uart_top/DUT/ASS... SVA		✓	Off	136	1	Unl...	1	100%		✓	0	0	0 ns	0
uart_top/DUT/ASS... SVA		✓	Off	136	1	Unl...	1	100%		✓	0	0	0 ns	0
uart_top/DUT/ASS... SVA		✓	Off	3	1	Unl...	1	100%		✓	0	0	0 ns	0
uart_top/DUT/ASS... SVA		✓	Off	15	1	Unl...	1	100%		✓	0	0	0 ns	0
uart_top/DUT/ASS... SVA		✓	Off	937	1	Unl...	1	100%		✓	0	0	0 ns	0
uart_top/DUT/ASS... SVA		✓	Off	11	1	Unl...	1	100%		✓	0	0	0 ns	0
uart_top/DUT/ASS... SVA		✓	Off	11	1	Unl...	1	100%		✓	0	0	0 ns	0
uart_top/DUT/ASS... SVA		✓	Off	22	1	Unl...	1	100%		✓	0	0	0 ns	0
uart_top/DUT/ASS... SVA		✓	Off	3	1	Unl...	1	100%		✓	0	0	0 ns	0
uart_top/DUT/ASS... SVA		✓	Off	14	1	Unl...	1	100%		✓	0	0	0 ns	0
uart_top/DUT/ASS... SVA		✓	Off	3	1	Unl...	1	100%		✓	0	0	0 ns	0
uart_top/DUT/ASS... SVA		✓	Off	15	1	Unl...	1	100%		✓	0	0	0 ns	0

Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment
/pack_coverage/u...		100.00%							
- TYPE UART_C...		100.00%	100	100.00...		✓			auto(1)
- CVP UART...		100.00%	100	100.00...		✓			
- CVP UART...		100.00%	100	100.00...		✓			
- CVP UART...		100.00%	100	100.00...		✓			
- CVP UART...		100.00%	100	100.00...		✓			
- CROSS UA...		100.00%	100	100.00...		✓			
- CROSS UA...		100.00%	100	100.00...		✓			

note when we add :

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

CROSS UA...	75.00%	100	75.00%		✓
- bin <au...	77	1	100.00...		✓
- bin <au...	144	1	100.00...		✓
- bin <au...	999	1	100.00...		✓
- bin <au...	0	1	0.00%		✓

```
=====
== 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       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
  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 Unit	Design UnitType	Lang	File(Line)	Hits	Status
/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 Count	Pass 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(145)          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
-----
```

```

Statement Coverage:          15 Covered
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 deasse...
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/G0 --
```

```

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/G0 --
```

Line	Item	Count	Source
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.PAK_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 لِإِنْسَانٍ إِلَّا مَا سَعَى *

[النجم: 39]

صدق الله العظيم