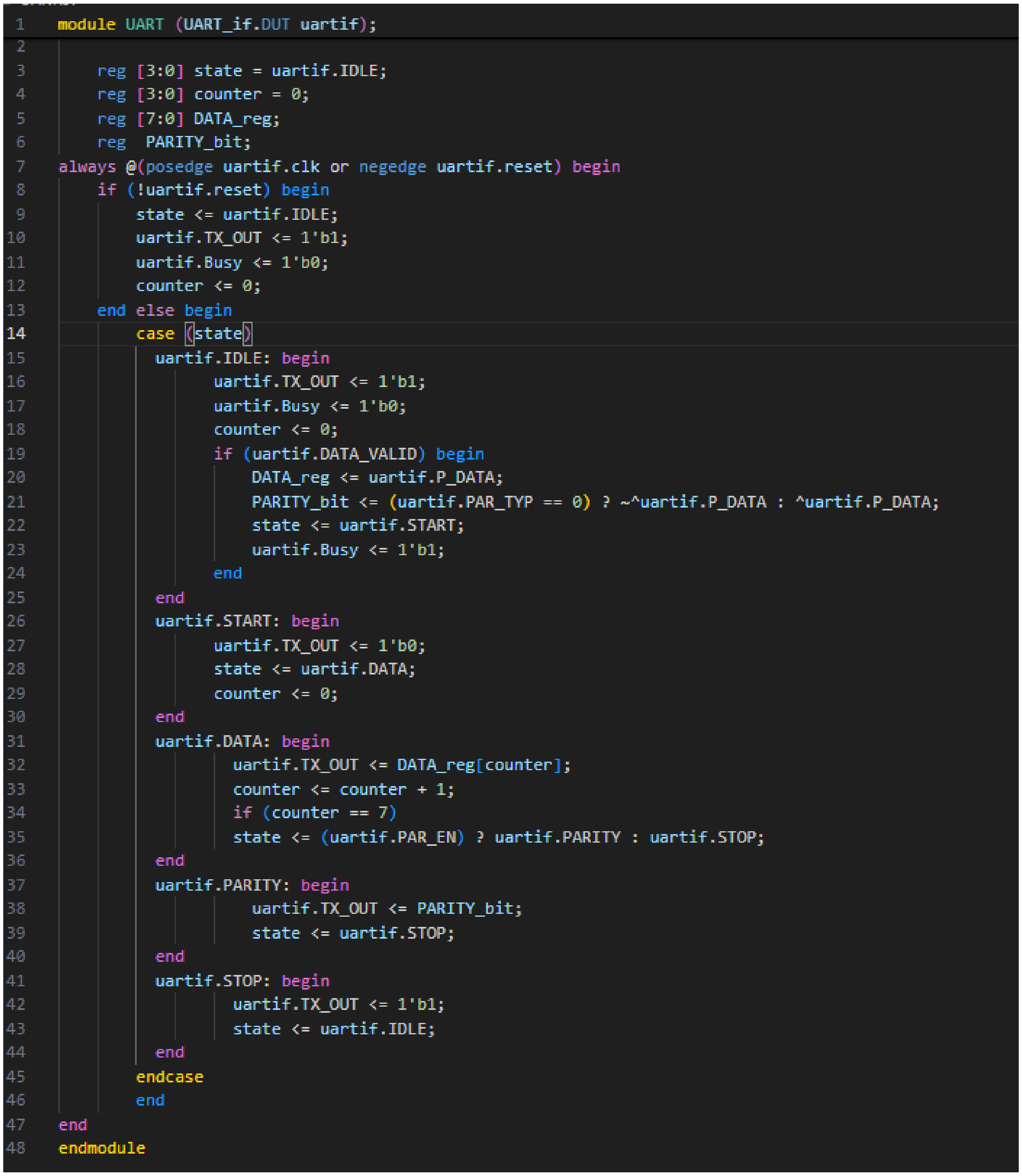
Momen Mostafa Mohamed Elzaghawy

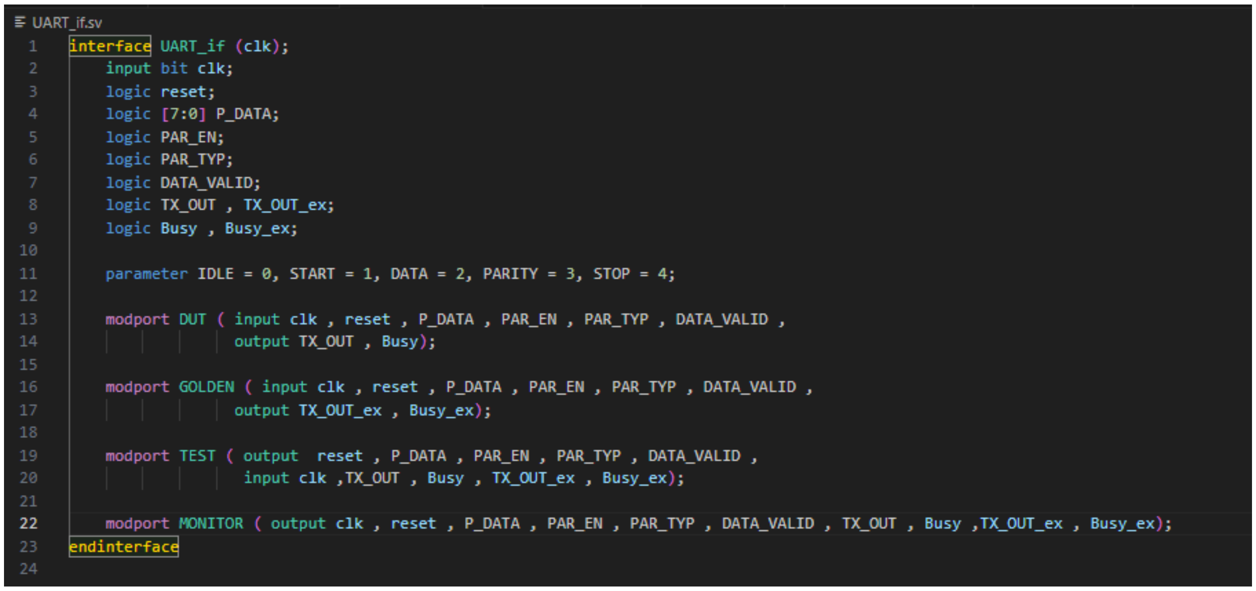
UART_TX SV

Design:



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

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

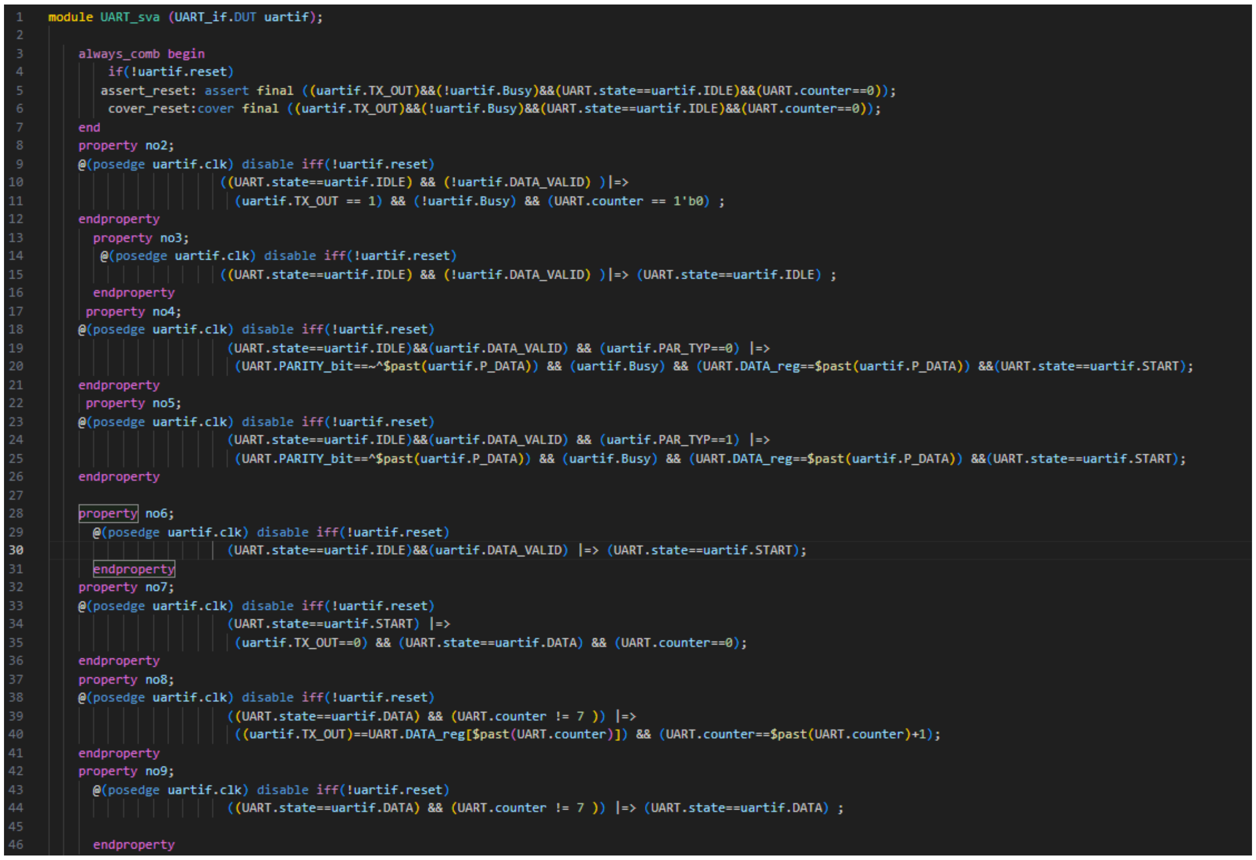
interface:



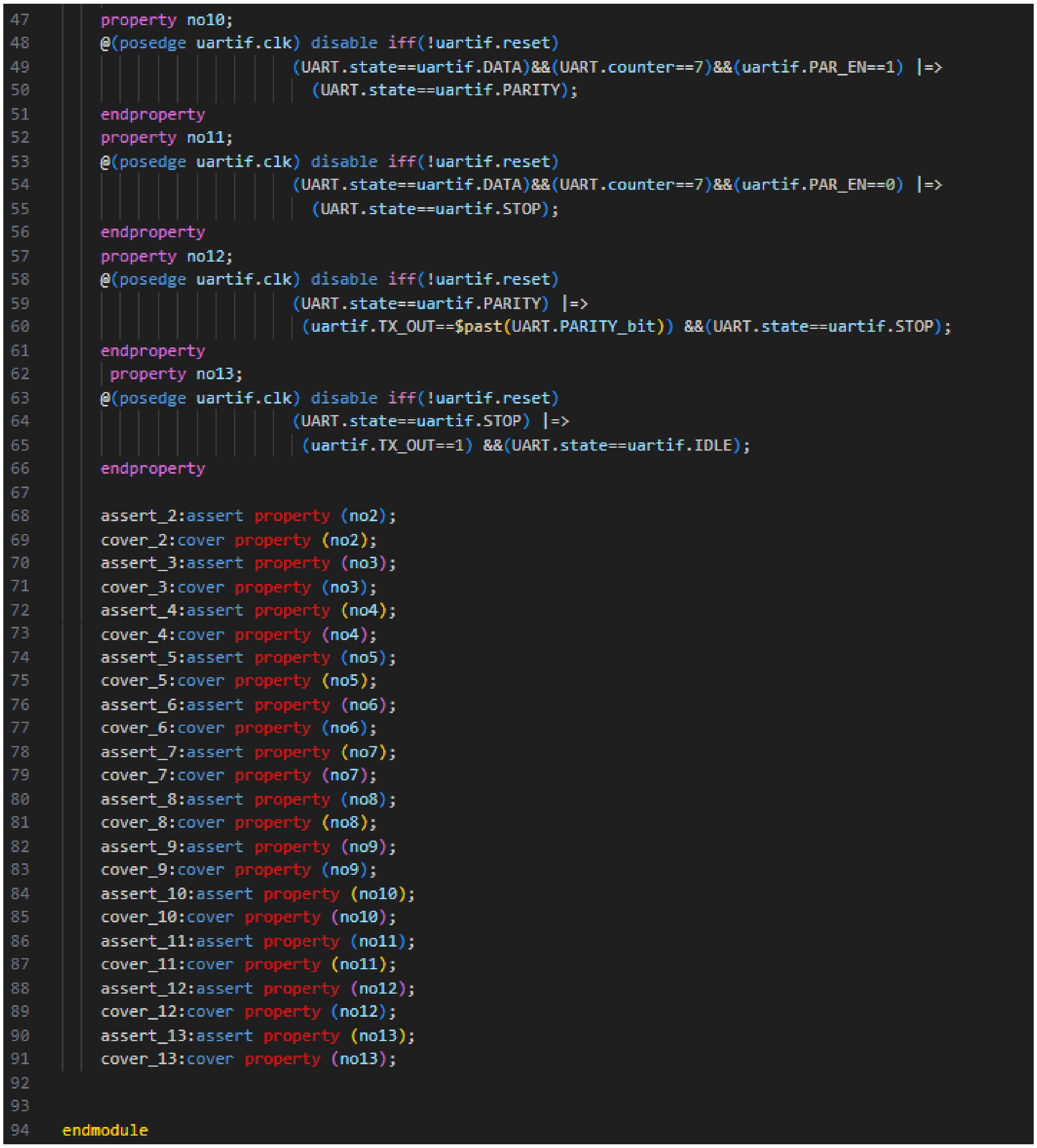
UART_if.sv

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

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

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

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

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

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

```

SVA:

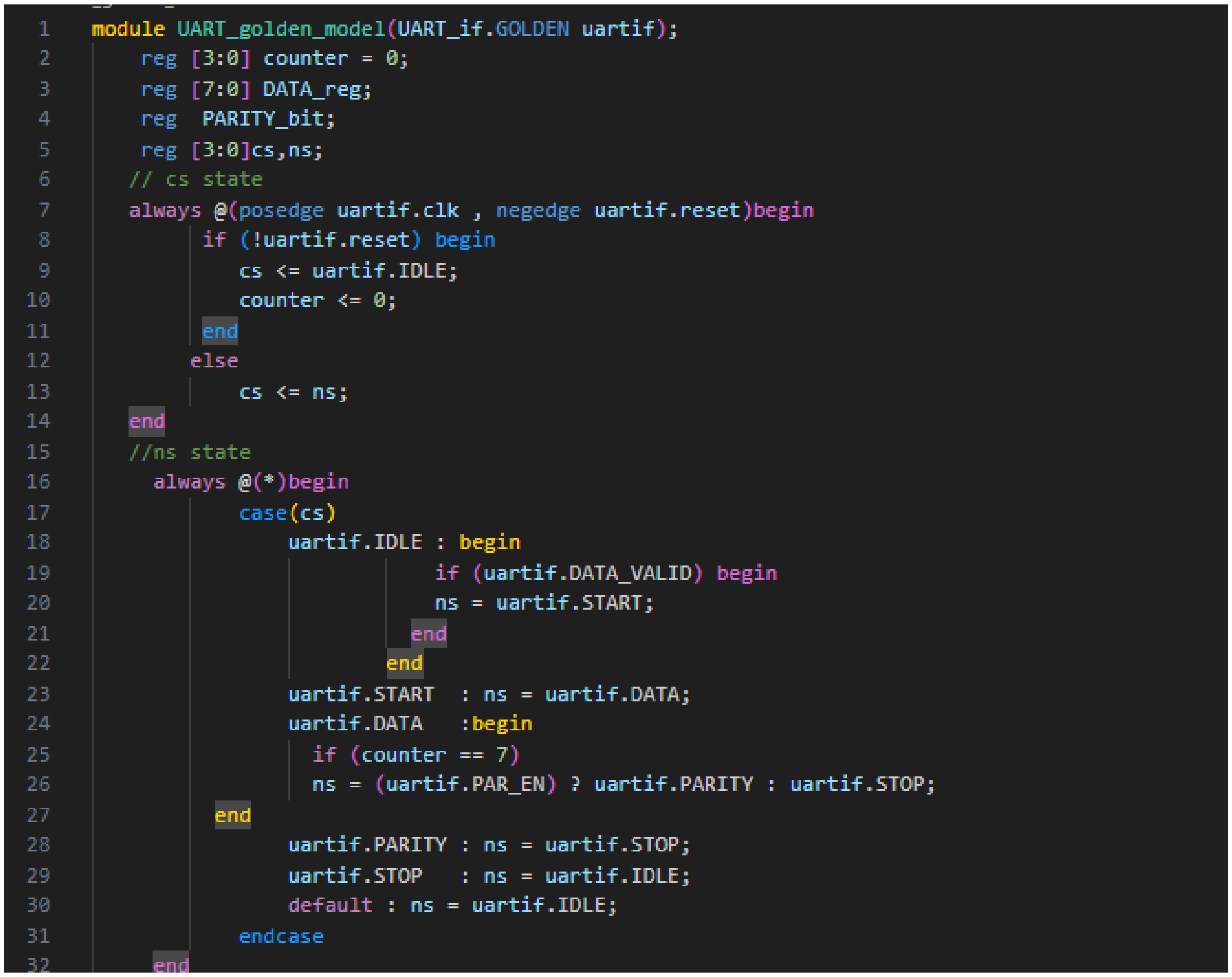


```
module UART_sva (UART_if.DUT uartif);

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

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

      endproperty
```

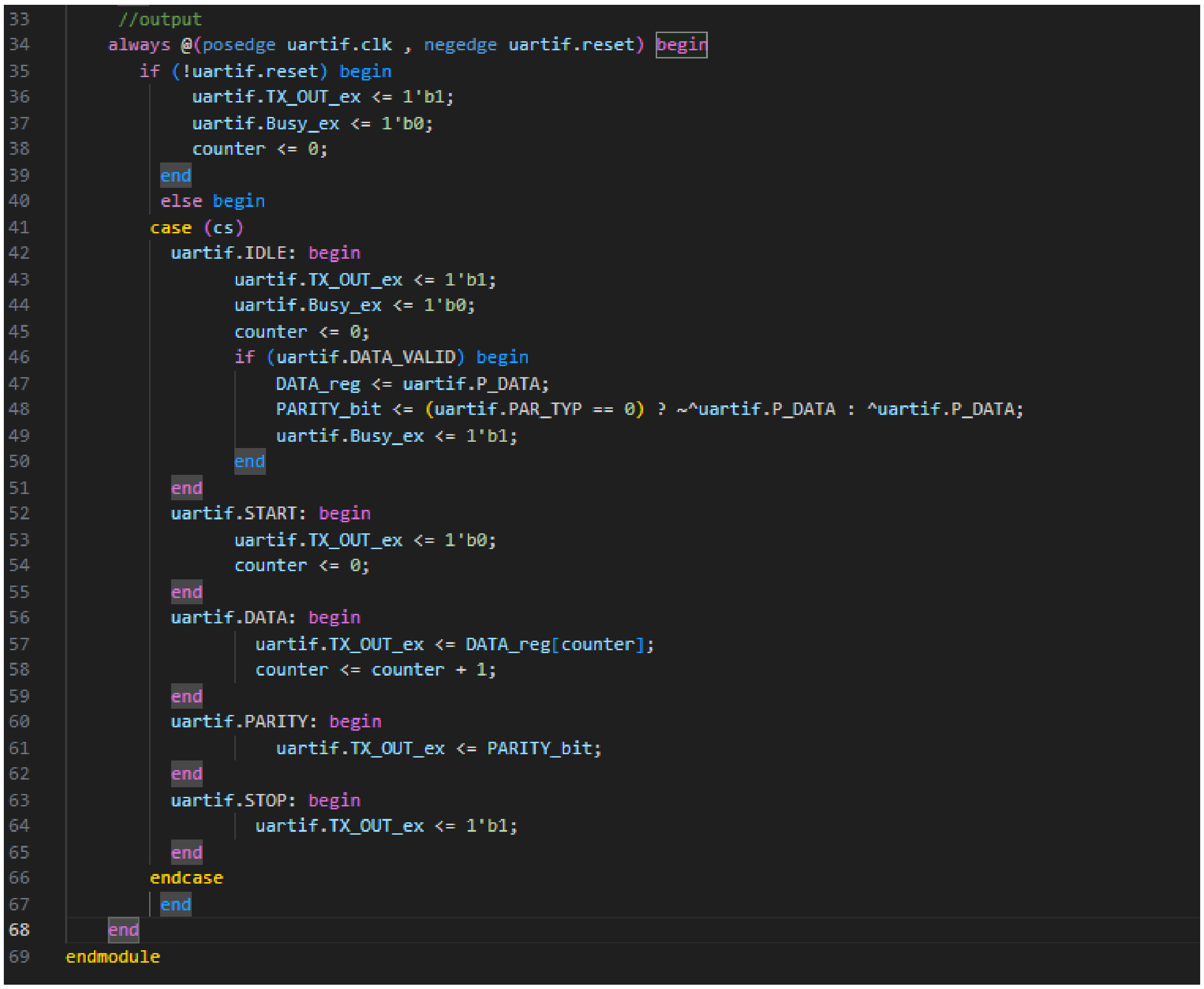


```
    property no10;
    @(posedge uartif.clk) disable iff(!uartif.reset)
                    (UART.state==uartif.DATA)&&(UART.counter==7)&&(uartif.PAR_EN==1) |=>
                      (UART.state==uartif.PARITY);
    endproperty
    property no11;
    @(posedge uartif.clk) disable iff(!uartif.reset)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

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


endmodule
```

Golden model:

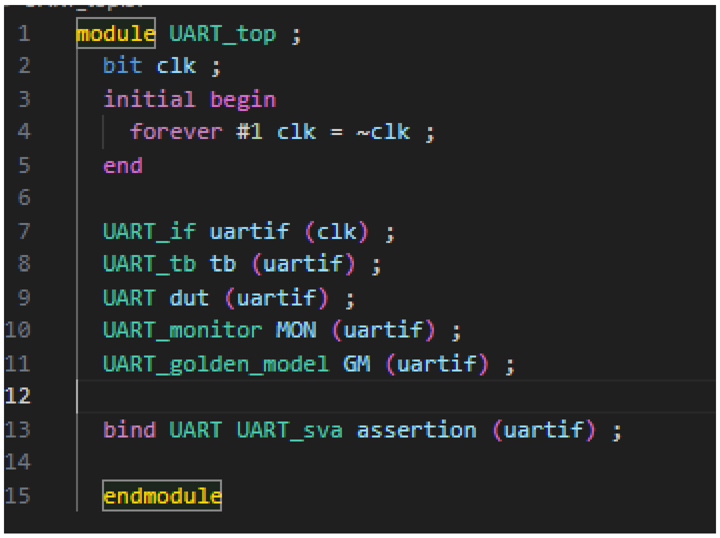


```
module UART_golden_model(UART_if.GOLDEN uartif);
    reg [3:0] counter = 0;
    reg [7:0] DATA_reg;
    reg  PARITY_bit;
    reg [3:0]cs,ns;
   // cs state
   always @(posedge uartif.clk , negedge uartif.reset)begin
         if (!uartif.reset) begin
            cs <= uartif.IDLE;
            counter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



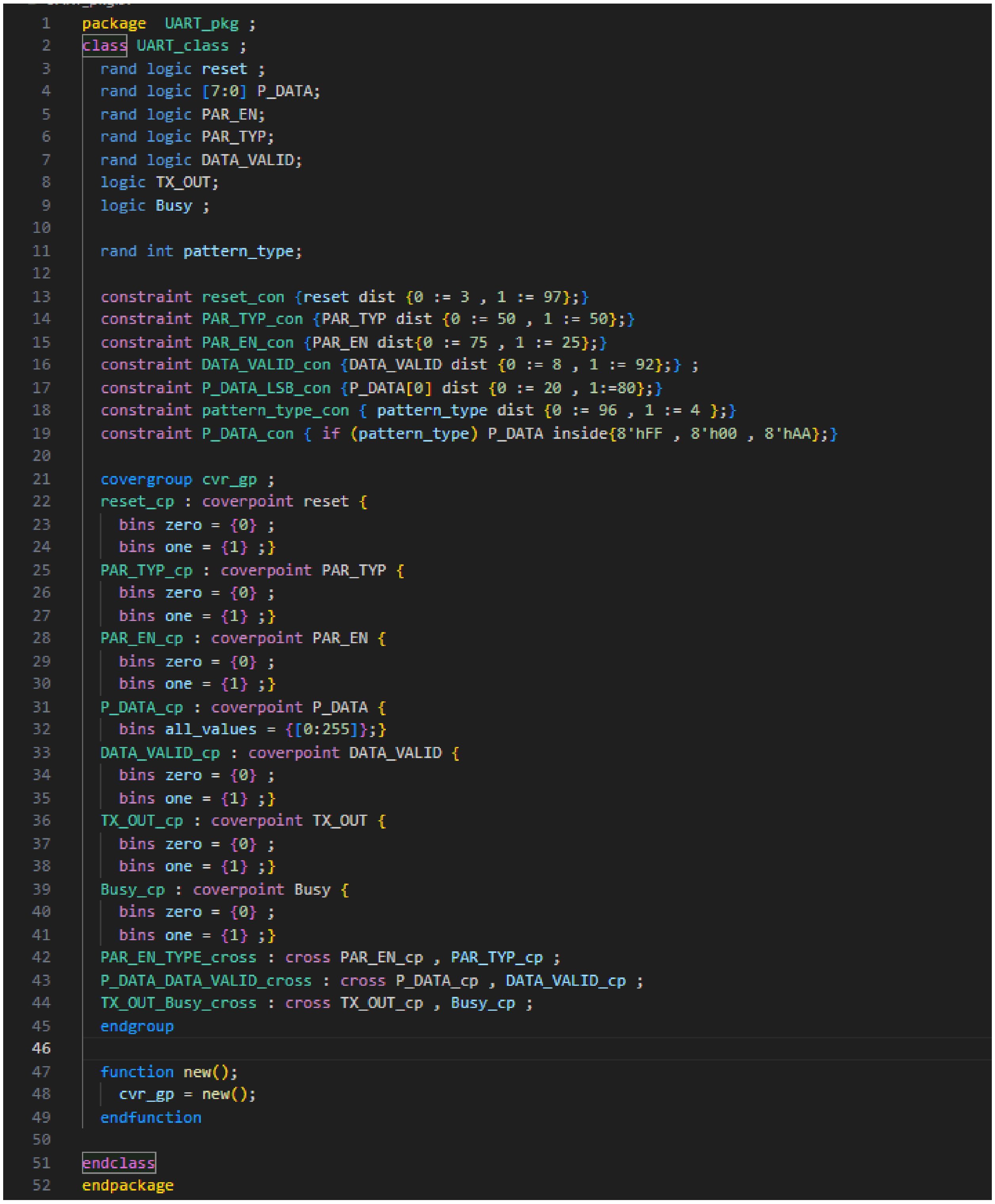
```
     //output
    always @(posedge uartif.clk , negedge uartif.reset) begin
       if (!uartif.reset) begin
            uartif.TX_OUT_ex <= 1'b1;
            uartif.Busy_ex <= 1'b0;
            counter <= 0;
         end
         else begin
        case (cs)
          uartif.IDLE: begin
                uartif.TX_OUT_ex <= 1'b1;
                uartif.Busy_ex <= 1'b0;
                counter <= 0;
                if (uartif.DATA_VALID) begin
                    DATA_reg <= uartif.P_DATA;
                    PARITY_bit <= (uartif.PAR_TYP == 0) ? ~^uartif.P_DATA : ^uartif.P_DATA;
                    uartif.Busy_ex <= 1'b1;
                end
          end
          uartif.START: begin
                uartif.TX_OUT_ex <= 1'b0;
                counter <= 0;
          end
          uartif.DATA: begin
                  uartif.TX_OUT_ex <= DATA_reg[counter];
                  counter <= counter + 1;
          end
          uartif.PARITY: begin
                    uartif.TX_OUT_ex <= PARITY_bit;
          end
          uartif.STOP: begin
                  uartif.TX_OUT_ex <= 1'b1;
          end
        endcase
         end
    end
endmodule
```

Top:



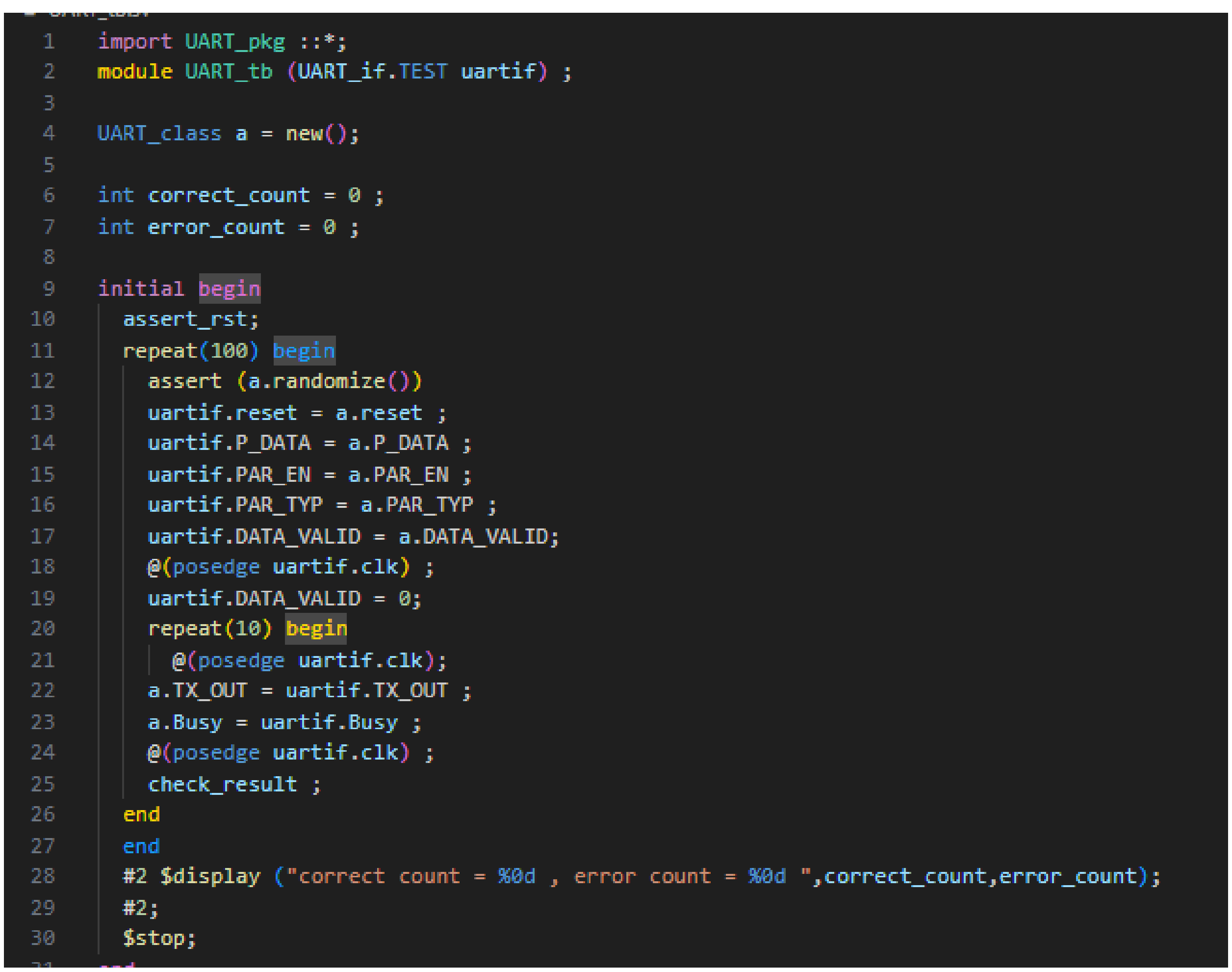
```
module UART_top ;
  bit clk ;
  initial begin
    forever #1 clk = ~clk ;
  end

  UART_if uartif (clk) ;
  UART_tb tb (uartif) ;
  UART dut (uartif) ;
  UART_monitor MON (uartif) ;
  UART_golden_model GM (uartif) ;

  bind UART UART_sva assertion (uartif) ;

  endmodule
```

Package:

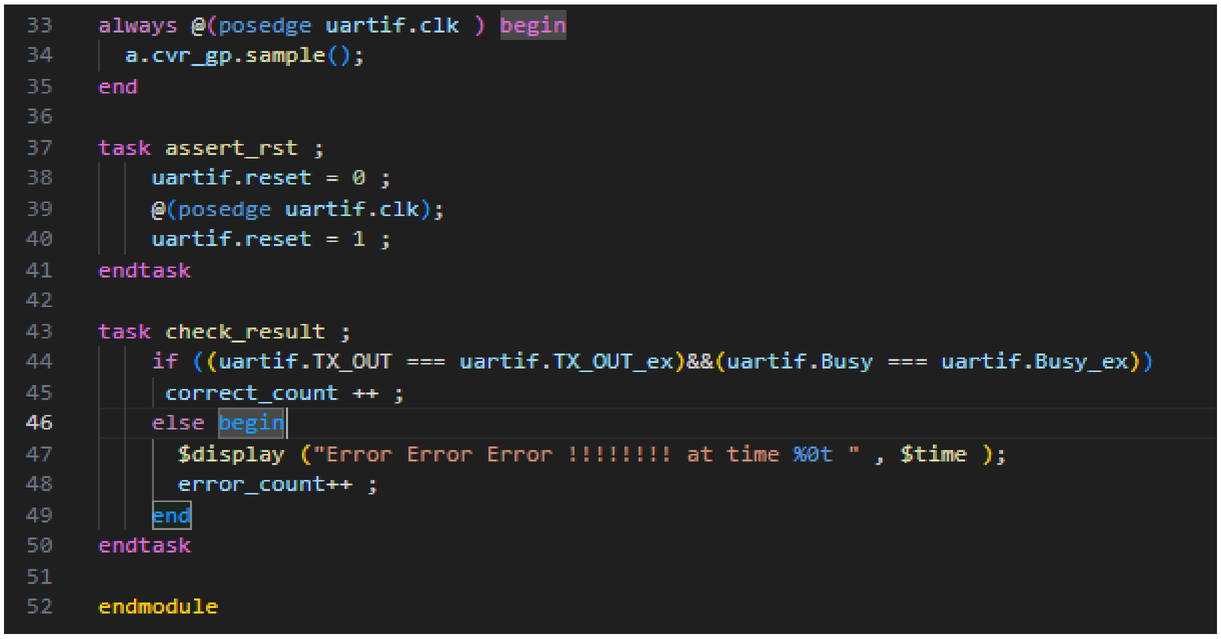


```
package  UART_pkg ;
class UART_class ;
  rand logic reset ;
  rand logic [7:0] P_DATA;
  rand logic PAR_EN;
  rand logic PAR_TYP;
  rand logic DATA_VALID;
  logic TX_OUT;
  logic Busy ;

  rand int pattern_type;

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

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

  function new();
    cvr_gp = new();
  endfunction

endclass
endpackage
```

Testbench:

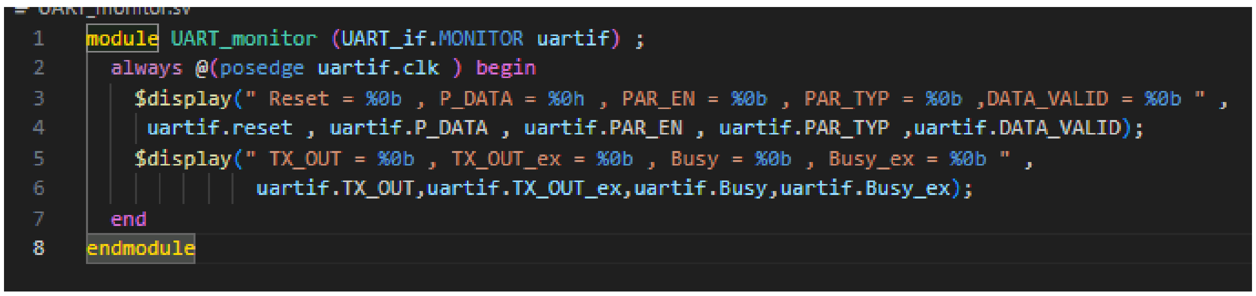


```
import UART_pkg ::*;
module UART_tb (UART_if.TEST uartif) ;

UART_class a = new();

int correct_count = 0 ;
int error_count = 0 ;

initial begin
  assert_rst;
  repeat(100) begin
    assert (a.randomize())
    uartif.reset = a.reset ;
    uartif.P_DATA = a.P_DATA ;
    uartif.PAR_EN = a.PAR_EN ;
    uartif.PAR_TYP = a.PAR_TYP ;
    uartif.DATA_VALID = a.DATA_VALID;
    @(posedge uartif.clk) ;
    uartif.DATA_VALID = 0;
    repeat(10) begin
      @(posedge uartif.clk);
    a.TX_OUT = uartif.TX_OUT ;
    a.Busy = uartif.Busy ;
    @(posedge uartif.clk) ;
    check_result ;
  end
  end
  #2 $display ("correct count = %0d , error count = %0d ",correct_count,error_count);
  #2;
  $stop;
```



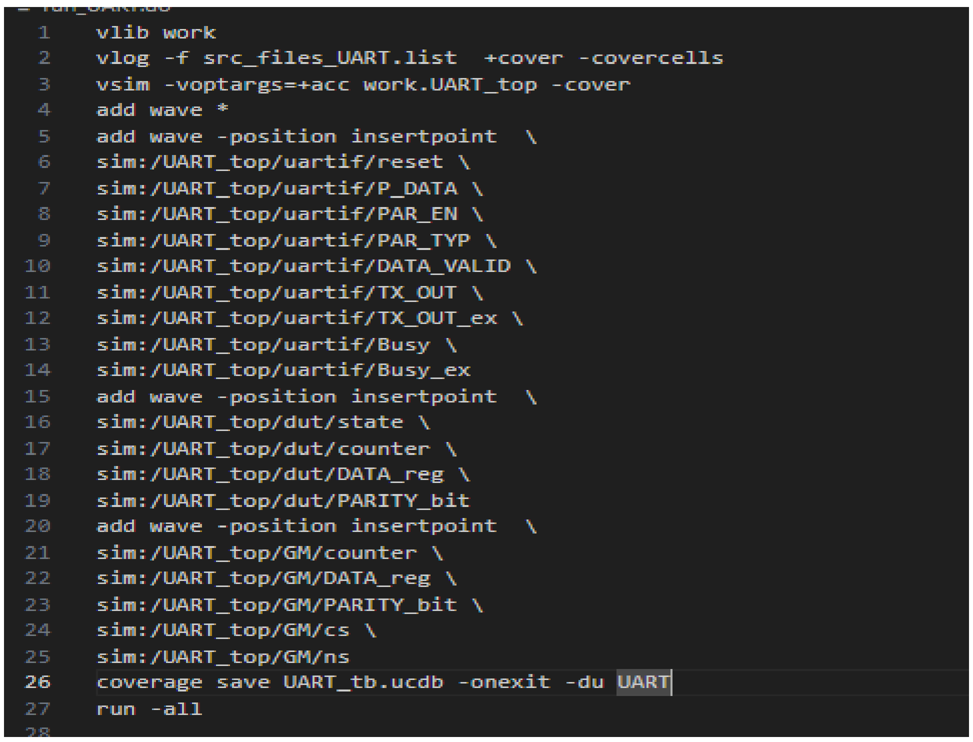
```
always @(posedge uartif.clk ) begin
  a.cvr_gp.sample();
end

task assert_rst ;
    uartif.reset = 0 ;
    @(posedge uartif.clk);
    uartif.reset = 1 ;
endtask

task check_result ;
    if ((uartif.TX_OUT === uartif.TX_OUT_ex)&&(uartif.Busy === uartif.Busy_ex))
     correct_count ++ ;
    else begin
      $display ("Error Error Error !!!!!!!! at time %0t " , $time );
      error_count++ ;
    end
endtask

endmodule
```

Monitor:



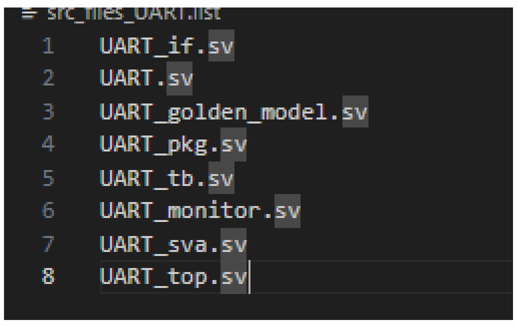
```
module UART_monitor (UART_if.MONITOR uartif) ;
  always @(posedge uartif.clk ) begin
    $display(" Reset = %0b , P_DATA = %0h , PAR_EN = %0b , PAR_TYP = %0b ,DATA_VALID = %0b " ,
     uartif.reset , uartif.P_DATA , uartif.PAR_EN , uartif.PAR_TYP ,uartif.DATA_VALID);
    $display(" TX_OUT = %0b , TX_OUT_ex = %0b , Busy = %0b , Busy_ex = %0b " ,
              uartif.TX_OUT,uartif.TX_OUT_ex,uartif.Busy,uartif.Busy_ex);
  end
endmodule
```

Do file:



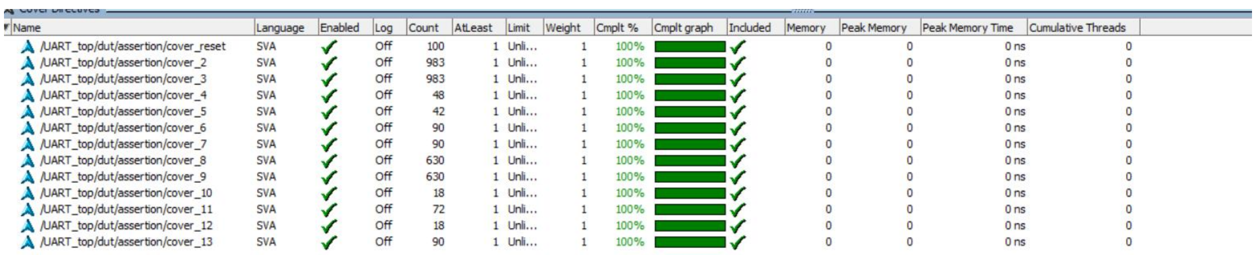
```
vlib work
vlog -f src_files_UART.list  +cover -covercells
vsim -voptargs=+acc work.UART_top -cover
add wave *
add wave -position insertpoint  \
sim:/UART_top/uartif/reset \
sim:/UART_top/uartif/P_DATA \
sim:/UART_top/uartif/PAR_EN \
sim:/UART_top/uartif/PAR_TYP \
sim:/UART_top/uartif/DATA_VALID \
sim:/UART_top/uartif/TX_OUT \
sim:/UART_top/uartif/TX_OUT_ex \
sim:/UART_top/uartif/Busy \
sim:/UART_top/uartif/Busy_ex
add wave -position insertpoint  \
sim:/UART_top/dut/state \
sim:/UART_top/dut/counter \
sim:/UART_top/dut/DATA_reg \
sim:/UART_top/dut/PARITY_bit
add wave -position insertpoint  \
sim:/UART_top/GM/counter \
sim:/UART_top/GM/DATA_reg \
sim:/UART_top/GM/PARITY_bit \
sim:/UART_top/GM/cs \
sim:/UART_top/GM/ns
coverage save UART_tb.ucdb -onexit -du UART
run -all
```

Src_files:

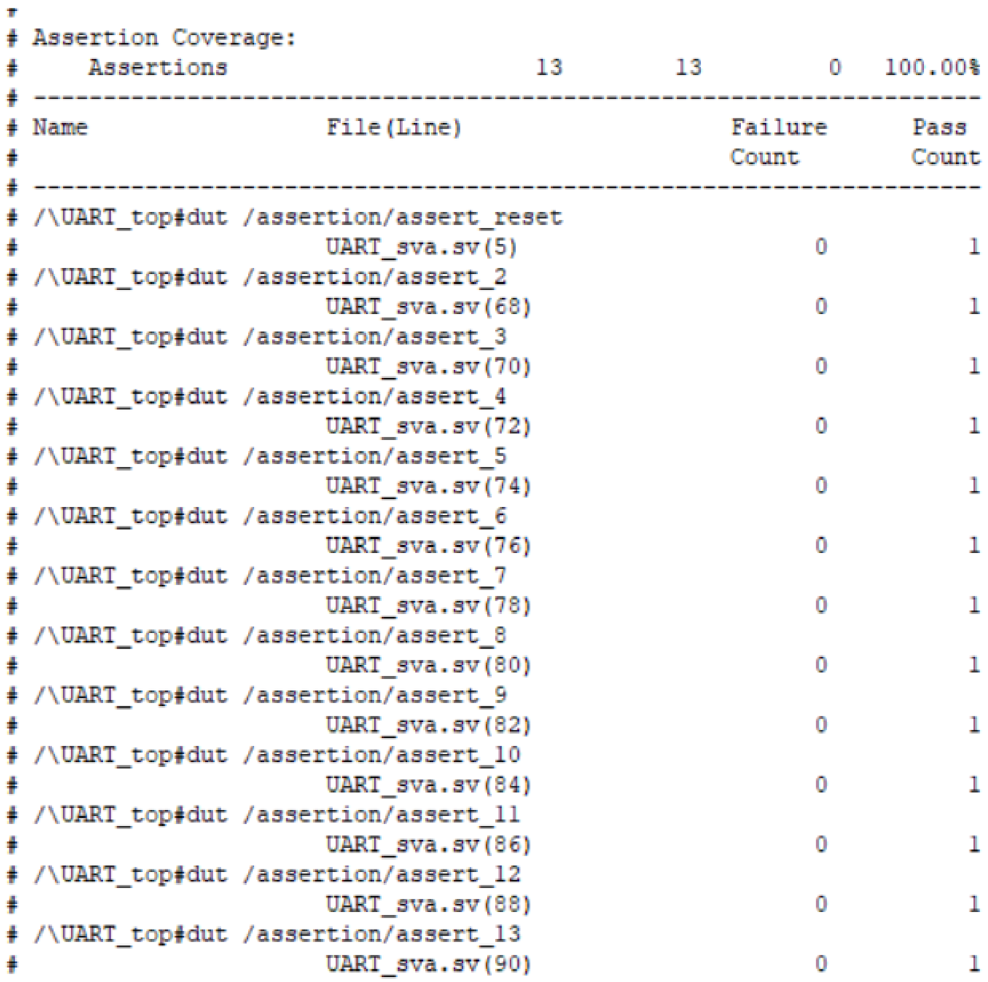


```
UART_if.sv
UART.sv
UART_golden_model.sv
UART_pkg.sv
UART_tb.sv
UART_monitor.sv
UART_sva.sv
UART_top.sv
```

SVA coverage:

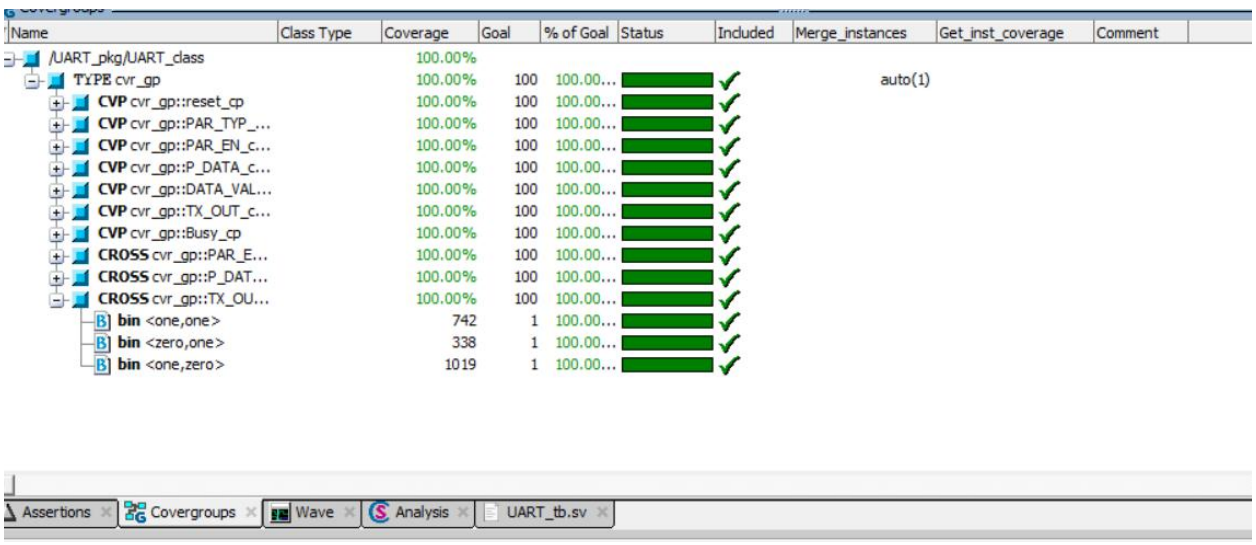


Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
/UART_top/dut/assertion/cover_reset	SVA	✓	Off	100	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/UART_top/dut/assertion/cover_2	SVA	✓	Off	983	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/UART_top/dut/assertion/cover_3	SVA	✓	Off	983	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/UART_top/dut/assertion/cover_4	SVA	✓	Off	48	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/UART_top/dut/assertion/cover_5	SVA	✓	Off	42	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/UART_top/dut/assertion/cover_6	SVA	✓	Off	90	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/UART_top/dut/assertion/cover_7	SVA	✓	Off	90	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/UART_top/dut/assertion/cover_8	SVA	✓	Off	630	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/UART_top/dut/assertion/cover_9	SVA	✓	Off	630	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/UART_top/dut/assertion/cover_10	SVA	✓	Off	18	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/UART_top/dut/assertion/cover_11	SVA	✓	Off	72	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/UART_top/dut/assertion/cover_12	SVA	✓	Off	18	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0
/UART_top/dut/assertion/cover_13	SVA	✓	Off	90	1	Unli...	1	100%	🟩	✓	0	0	0 ns	0



```
#
# Assertion Coverage:
#     Assertions                        13         13          0   100.00%
# --------------------------------------------------------------------
# Name                     File(Line)                   Failure      Pass
#                                                       Count        Count
# --------------------------------------------------------------------
# /\UART_top#dut /assertion/assert_reset
#                          UART_sva.sv(5)                     0          1
# /\UART_top#dut /assertion/assert_2
#                          UART_sva.sv(68)                    0          1
# /\UART_top#dut /assertion/assert_3
#                          UART_sva.sv(70)                    0          1
# /\UART_top#dut /assertion/assert_4
#                          UART_sva.sv(72)                    0          1
# /\UART_top#dut /assertion/assert_5
#                          UART_sva.sv(74)                    0          1
# /\UART_top#dut /assertion/assert_6
#                          UART_sva.sv(76)                    0          1
# /\UART_top#dut /assertion/assert_7
#                          UART_sva.sv(78)                    0          1
# /\UART_top#dut /assertion/assert_8
#                          UART_sva.sv(80)                    0          1
# /\UART_top#dut /assertion/assert_9
#                          UART_sva.sv(82)                    0          1
# /\UART_top#dut /assertion/assert_10
#                          UART_sva.sv(84)                    0          1
# /\UART_top#dut /assertion/assert_11
#                          UART_sva.sv(86)                    0          1
# /\UART_top#dut /assertion/assert_12
#                          UART_sva.sv(88)                    0          1
# /\UART_top#dut /assertion/assert_13
#                          UART_sva.sv(90)                    0          1
```

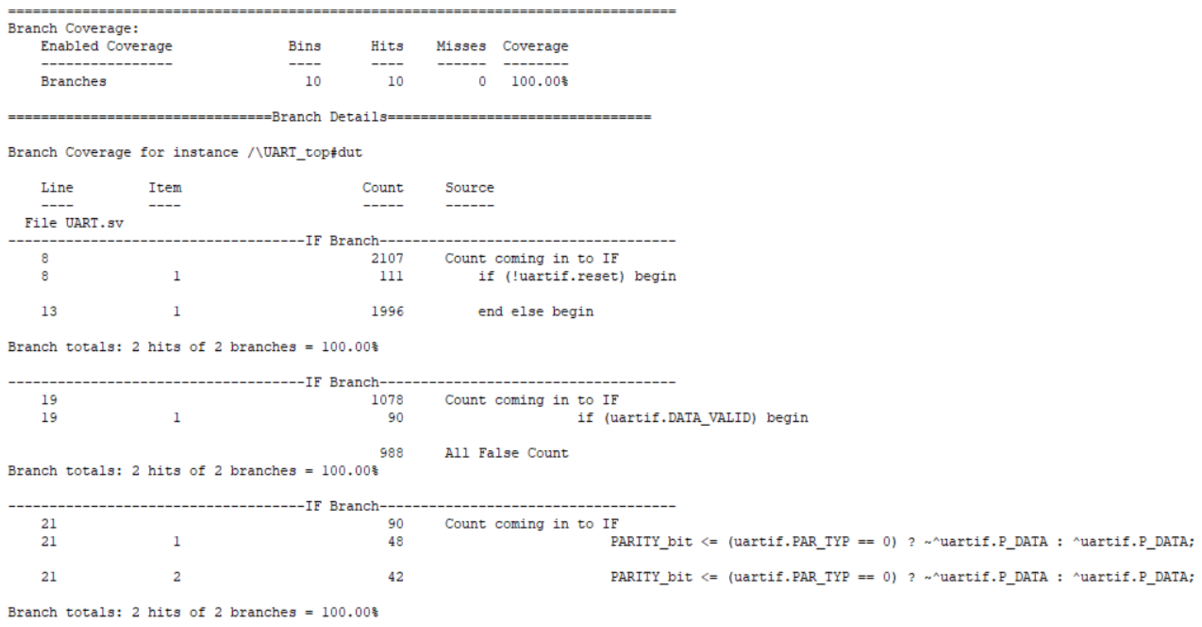
Functional coverage:



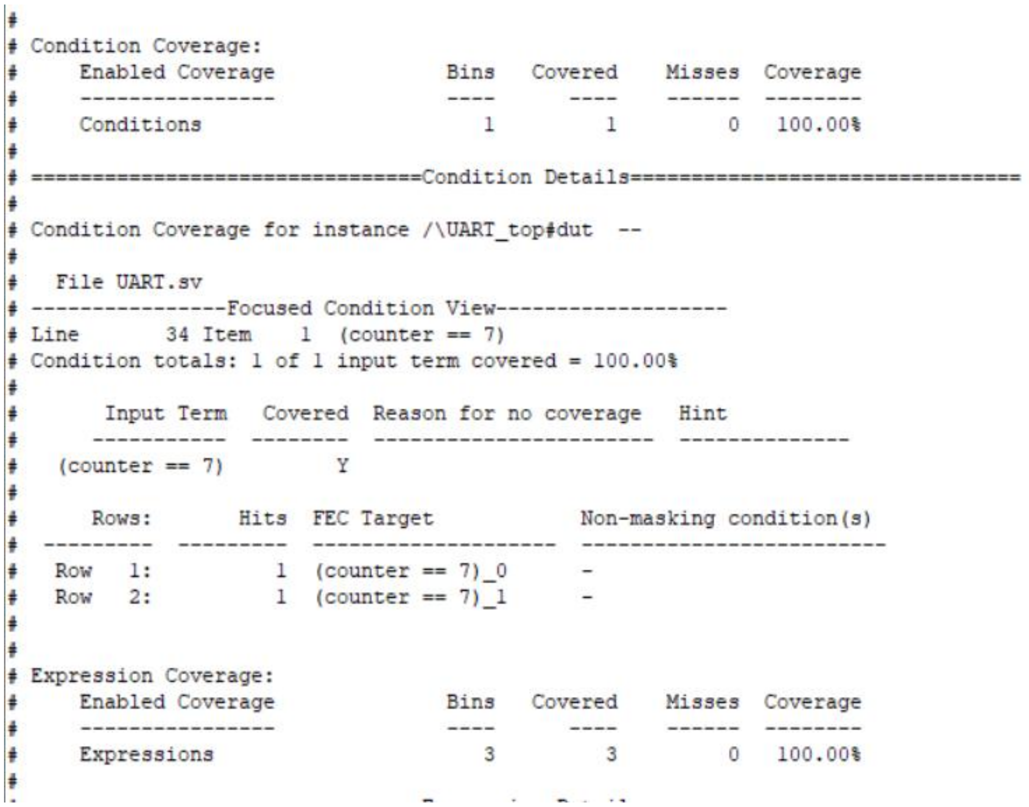
Name	Class Type	Coverage	Goal	% of Goal	Status	Included	Merge_instances	Get_inst_coverage	Comment
/UART_pkg/UART_class		100.00%							
TYPE cvr_gp		100.00%	100	100.00...	🟩	✓	auto(1)		
CVP cvr_gp::reset_cp		100.00%	100	100.00...	🟩	✓			
CVP cvr_gp::PAR_TYP_...		100.00%	100	100.00...	🟩	✓			
CVP cvr_gp::PAR_EN_c...		100.00%	100	100.00...	🟩	✓			
CVP cvr_gp::P_DATA_c...		100.00%	100	100.00...	🟩	✓			
CVP cvr_gp::DATA_VAL...		100.00%	100	100.00...	🟩	✓			
CVP cvr_gp::TX_OUT_c...		100.00%	100	100.00...	🟩	✓			
CVP cvr_gp::Busy_cp		100.00%	100	100.00...	🟩	✓			
CROSS cvr_gp::PAR_E...		100.00%	100	100.00...	🟩	✓			
CROSS cvr_gp::P_DAT...		100.00%	100	100.00...	🟩	✓			
CROSS cvr_gp::TX_OU...		100.00%	100	100.00...	🟩	✓			
bin <one,one>		742	1	100.00...	🟩	✓			
bin <zero,one>		338	1	100.00...	🟩	✓			
bin <one,zero>		1019	1	100.00...	🟩	✓			

I excluded bin zero,zero in cross tx_out ,busy because they can't be zeros in the same clock cycle

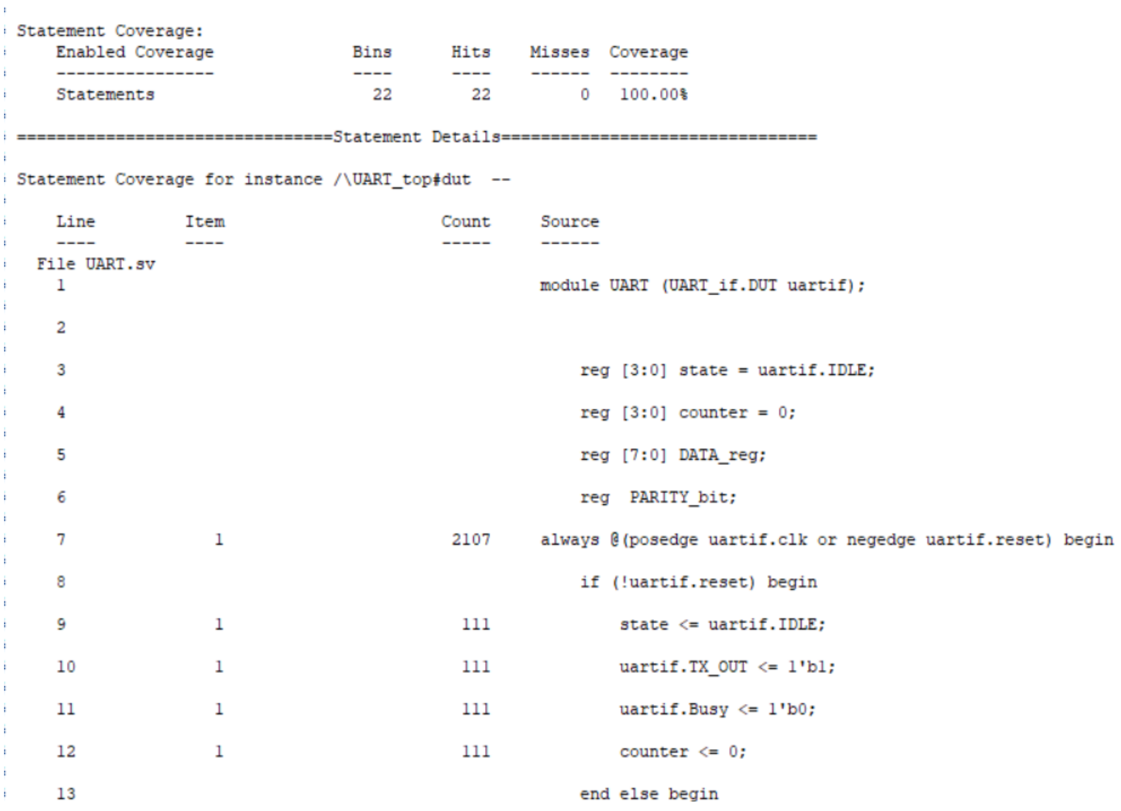
Code coverage:



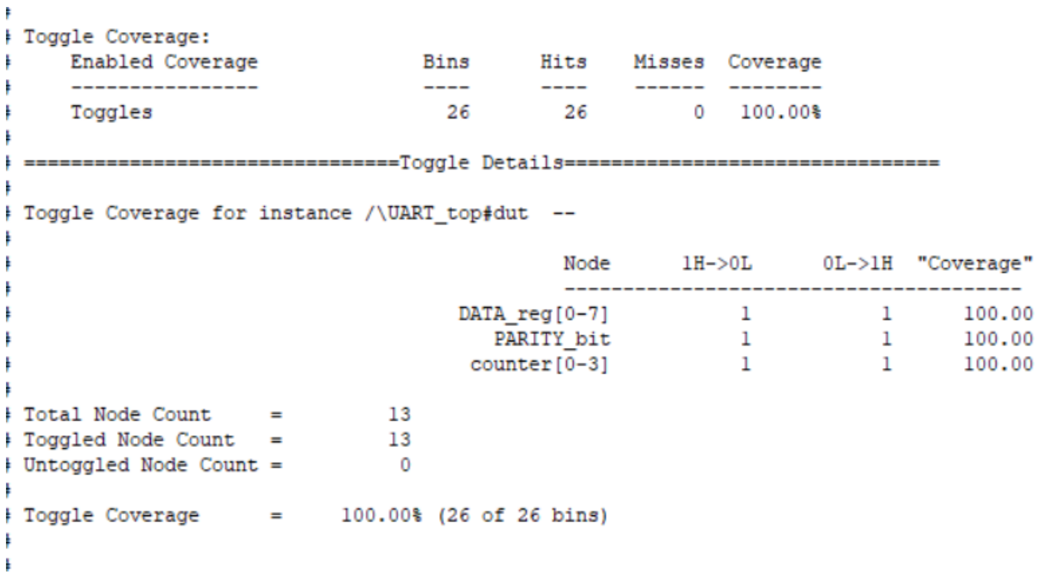
```
=================================================================================
Branch Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        10        10         0   100.00%

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

Branch Coverage for instance /\UART_top#dut

    Line         Item                      Count     Source
    ----         ----                      -----     ------
  File UART.sv
------------------------------------IF Branch------------------------------------
    8                                       2107     Count coming in to IF
    8               1                        111         if (!uartif.reset) begin

    13              1                       1996         end else begin

Branch totals: 2 hits of 2 branches = 100.00%

------------------------------------IF Branch------------------------------------
    19                                      1078     Count coming in to IF
    19              1                         90                 if (uartif.DATA_VALID) begin

                                             988     All False Count
Branch totals: 2 hits of 2 branches = 100.00%

------------------------------------IF Branch------------------------------------
    21                                        90     Count coming in to IF
    21              1                         48                     PARITY_bit <= (uartif.PAR_TYP == 0) ? ~^uartif.P_DATA : ^uartif.P_DATA;

    21              2                         42                     PARITY_bit <= (uartif.PAR_TYP == 0) ? ~^uartif.P_DATA : ^uartif.P_DATA;

Branch totals: 2 hits of 2 branches = 100.00%
```



```
#
# Condition Coverage:
#     Enabled Coverage              Bins   Covered    Misses  Coverage
#     ----------------              ----      ----    ------  --------
#     Conditions                       1         1         0   100.00%
#
# ================================Condition Details================================
#
# Condition Coverage for instance /\UART_top#dut  --
#
#   File UART.sv
# ----------------Focused Condition View-------------------
# Line       34 Item    1  (counter == 7)
# Condition totals: 1 of 1 input term covered = 100.00%
#
#       Input Term   Covered  Reason for no coverage   Hint
#      -----------  --------  -----------------------  --------------
#   (counter == 7)         Y
#
#      Rows:       Hits  FEC Target            Non-masking condition(s)
#  ---------  ---------  --------------------  -------------------------
#   Row   1:          1  (counter == 7)_0      -
#   Row   2:          1  (counter == 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#dut  --

    Line         Item                      Count     Source
    ----         ----                      -----     ------
  File UART.sv
    1                                                module UART (UART_if.DUT uartif);

    2

    3                                                    reg [3:0] state = uartif.IDLE;

    4                                                    reg [3:0] counter = 0;

    5                                                    reg [7:0] DATA_reg;

    6                                                    reg  PARITY_bit;

    7                1                      2107     always @(posedge uartif.clk or negedge uartif.reset) begin

    8                                                    if (!uartif.reset) begin

    9                1                       111             state <= uartif.IDLE;

    10               1                       111             uartif.TX_OUT <= 1'b1;

    11               1                       111             uartif.Busy <= 1'b0;

    12               1                       111             counter <= 0;

    13                                                   end else begin
```

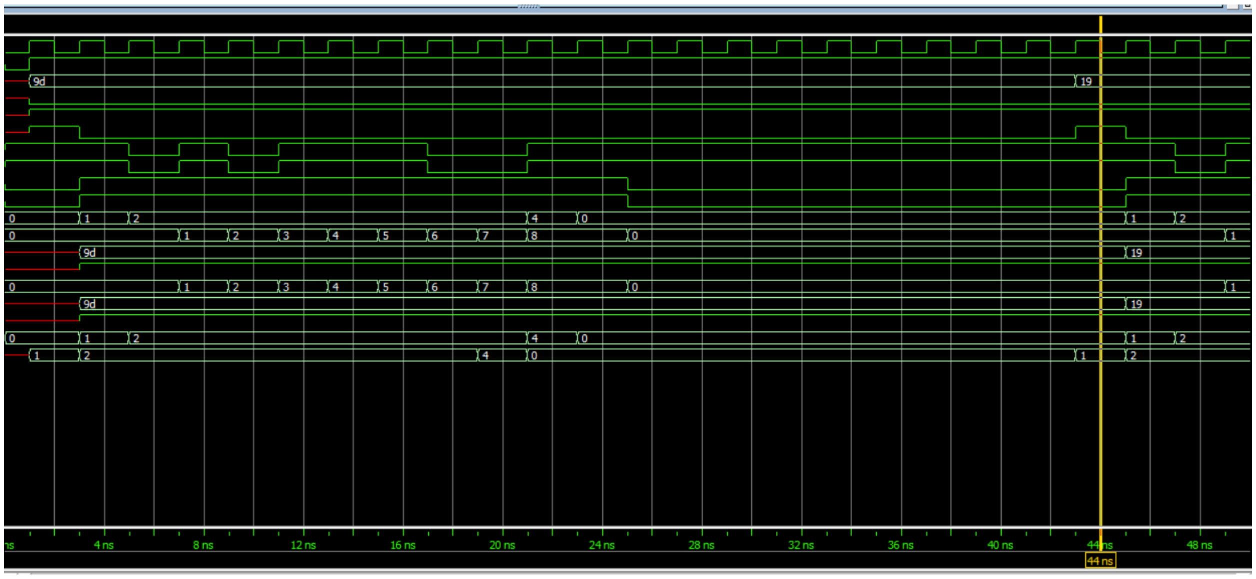


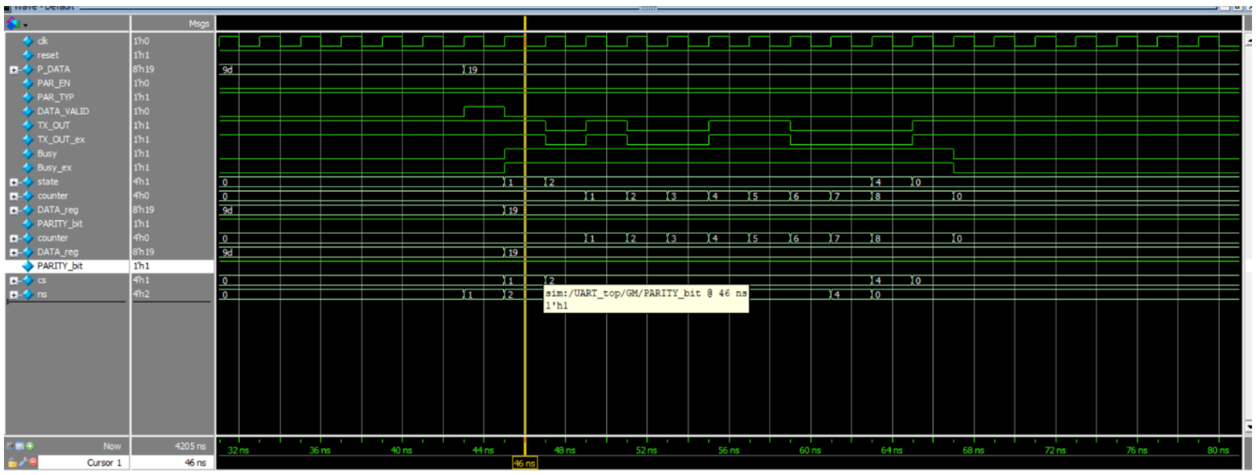
```
#
# Toggle Coverage:
#     Enabled Coverage              Bins      Hits    Misses  Coverage
#     ----------------              ----      ----    ------  --------
#     Toggles                         26        26         0   100.00%
#
# ================================Toggle Details================================
#
# Toggle Coverage for instance /\UART_top#dut  --
#
#                                               Node      1H->0L      0L->1H   "Coverage"
#                                               ---------------------------------------
#                                       DATA_reg[0-7]           1           1       100.00
#                                          PARITY_bit           1           1       100.00
#                                        counter[0-3]           1           1       100.00
#
# Total Node Count     =         13
# Toggled Node Count   =         13
# Untoggled Node Count =          0
#
# Toggle Coverage      =     100.00% (26 of 26 bins)
#
#
```

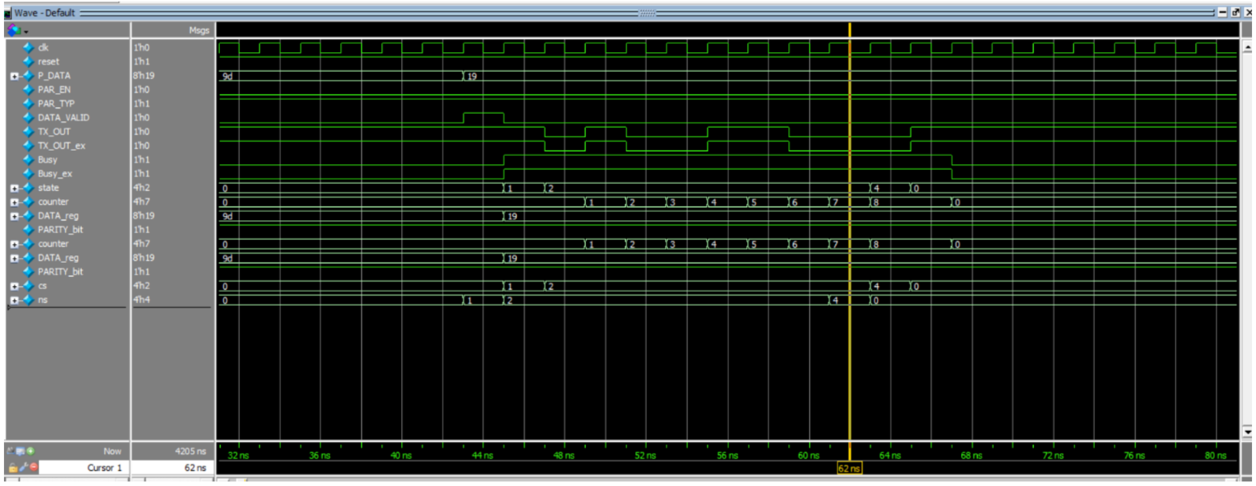
I excluded state in branch and toggle because we have only 5 states which only needs [2:0] state and not [3:0].

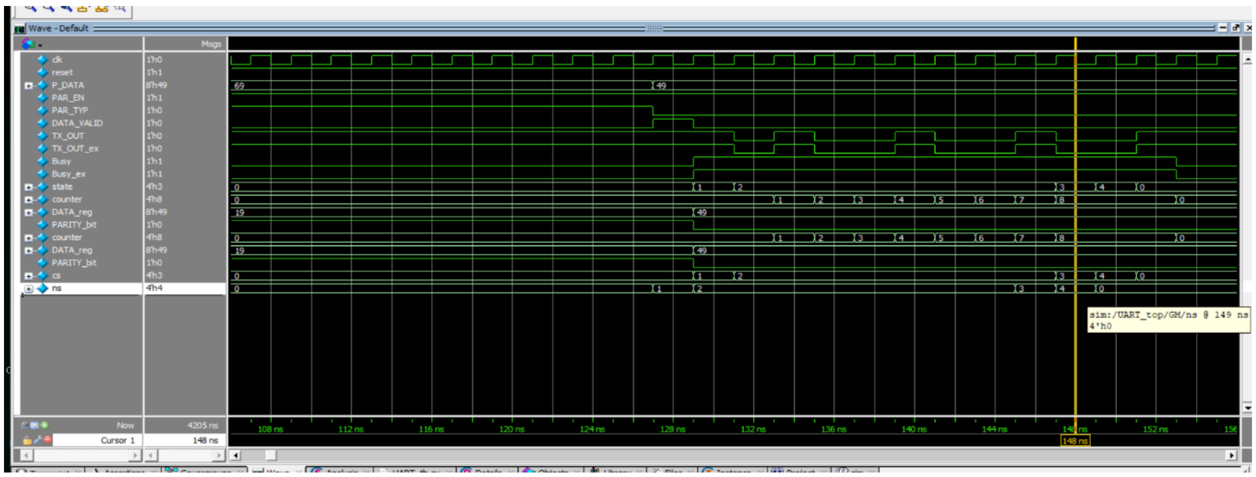
It can be considered as a bug in the design.

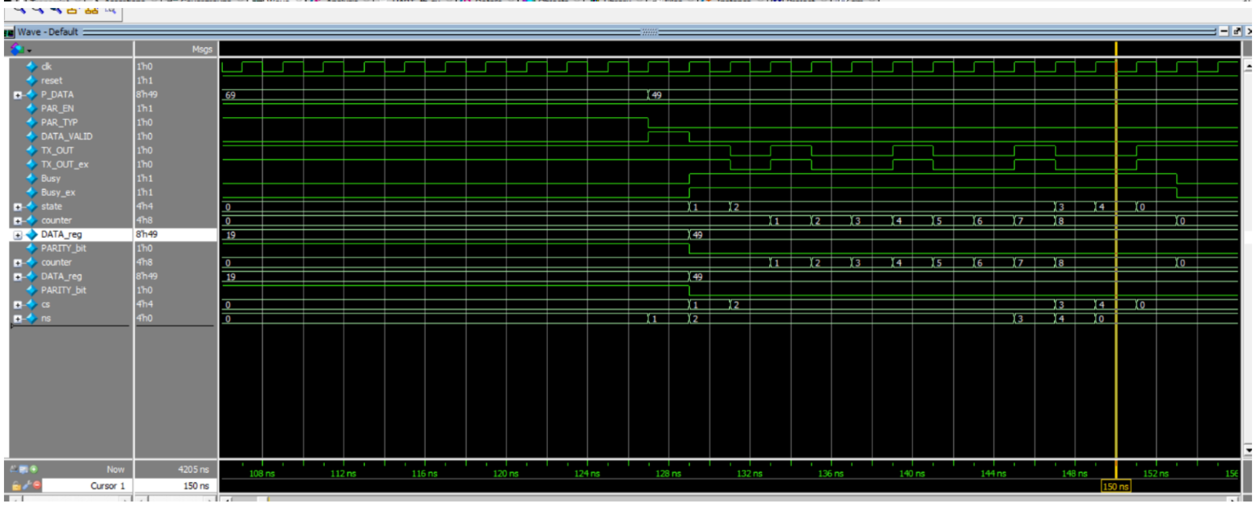
Output of all states and transcribt:




```
Msgs
clk          1'h0
reset        1'h1
P_DATA       8'h19
PAR_EN       1'h0
PAR_TYP      1'h1
DATA_VALID   1'h0
TX_OUT       1'h1
TX_OUT_ex    1'h1
Busy         1'h1
Busy_ex      1'h1
state        4'h1
counter      4'h0
DATA_reg     8'h19
PARITY_bit   1'h1
counter      4'h0
DATA_reg     8'h19
PARITY_bit   1'h1
cs           4'h1
ns           4'h2
Now          4205 ns
Cursor 1     46 ns
```
```
sim:/UART_top/GM/PARITY_bit @ 46 ns
1'h1
```


```
Wave - Default
Msgs
clk          1'h0
reset        1'h1
P_DATA       8'h19
PAR_EN       1'h0
PAR_TYP      1'h1
DATA_VALID   1'h0
TX_OUT       1'h0
TX_OUT_ex    1'h0
Busy         1'h1
Busy_ex      1'h1
state        4'h2
counter      4'h7
DATA_reg     8'h19
PARITY_bit   1'h1
counter      4'h7
DATA_reg     8'h19
PARITY_bit   1'h1
cs           4'h2
ns           4'h4
Now          4205 ns
Cursor 1     62 ns
```





```
                     Busy = 0b0 , Busy_ex = 0b0
 correct count = 1000 , error count = 0
  Reset = 0b1 , P_DATA = 0he4 , PAR_EN = 0b0 , PAR_TYP = 0b0 ,

      DATA_VALID = 0b0
 OUTPUT : TX_OUT = 0b1 , TX_OUT_ex = 0b1 ,

                     Busy = 0b0 , Busy_ex = 0b0
 ** Note: $stop    : UART_tb.sv(30)
    Time: 4205 ns  Iteration: 0  Instance: /UART_top/tb
 Break in Module UART_tb at UART_tb.sv line 30
```