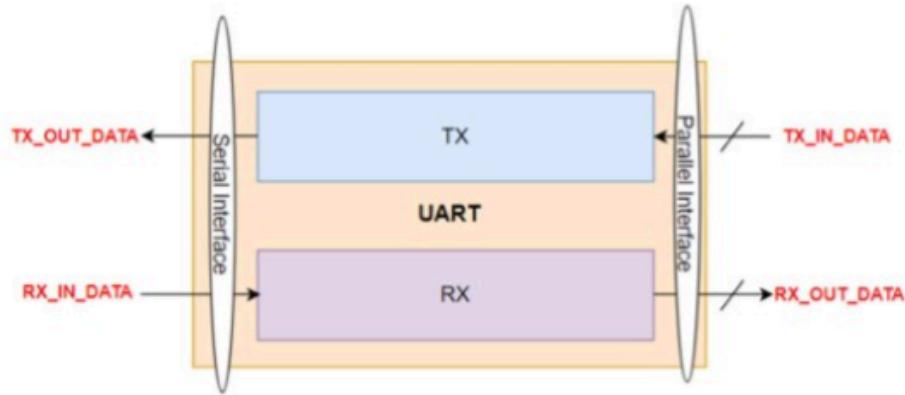


Abdalrahman Taha Mohammed

FULL_UART Using SV

Design :



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 TX:

```
module uart_tx (uart_if.TX u_if);

    reg [2:0] state = u_if.IDLE;
    reg [3:0] counter = 0;
    reg [7:0] DATA_reg;
    reg PARITY_bit;

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

golden model tx :

```
module uart_golden_model_tx (uart_if.GM_tx u_if);
reg [3:0] counter = 0;
reg [7:0] DATA_reg;
reg PARITY_bit;
reg [3:0] cs = u_if.IDLE, ns = u_if.IDLE;

// State memory
always @(posedge u_if.clk or negedge u_if.reset) begin
    if (!u_if.reset) begin
        cs <= u_if.IDLE;
        ns <= u_if.IDLE;
    end
    else begin
        cs <= ns;
    end
end

// Next state Logic
always @(*) begin
    case(cs)
        u_if.IDLE: begin
            if (u_if.DATA_VALID ) begin
                ns = u_if.START;
            end
        end
        u_if.START: ns = u_if.DATA;
        u_if.DATA: begin
            if (counter == 7) begin
                ns = (u_if.PAR_EN) ? u_if.PARITY : u_if.STOP;
            end
        end
        u_if.PARITY: ns = u_if.STOP;
        u_if.STOP: ns = u_if.IDLE;
        default: ns = u_if.IDLE;
    endcase
end

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

RTL RX :

```
module uart_rx (uart_if.RX u_if);

reg [3:0] counter = 0;
reg [7:0] DATA_reg;
reg [3:0] cs = u_if.IDLE;
reg [3:0] ns = u_if.IDLE;

// State memory
always @(posedge u_if.clk or negedge u_if.reset) begin
    if (!u_if.reset) begin
        cs <= u_if.IDLE;
    end
    else begin
        cs <= ns;
    end
end

// Next state Logic
always @(*) begin
    case(cs)
        u_if.IDLE: begin
            if (u_if.Busy /*&& u_if.TX_OUT*/) begin
                ns = u_if.START;
            end
        end
        u_if.START: ns = u_if.DATA;
        u_if.DATA: begin
            if (counter == 7) begin
                ns = (u_if.PAR_EN) ? u_if.PARITY : u_if.STOP;
            end
        end
        u_if.PARITY: ns = u_if.STOP;
        u_if.STOP: ns = u_if.IDLE;
        default: ns = u_if.IDLE;
    endcase
end

// Output logic
always @(posedge u_if.clk or negedge u_if.reset) begin
    if (!u_if.reset) begin
        counter <= 0;
        DATA_reg <= 0;
        u_if.RX_DONE <= 0;
        u_if.PAR_OUT <= 0;
    end
    else begin
        case (cs)
            u_if.IDLE: begin
                DATA_reg <= 0;
                counter <= 0;
                u_if.PAR_OUT <= 0;
                u_if.RX_DONE <= 0;
            end
            u_if.START: begin
                counter <= 0;
            end
            u_if.DATA: begin
                DATA_reg[counter] <= u_if.TX_OUT;
                counter <= counter + 1;
            end
            u_if.PARITY: begin
                u_if.PAR_OUT <= u_if.TX_OUT;
            end
            u_if.STOP: begin
                u_if.P_DATA_OUT <= DATA_reg;
                u_if.RX_DONE <= 1'b1 ;
            end
        endcase
    end
end
endmodule
```

golden model rx :

```
module uart_golden_model_rx (uart_if.GM_RX u_if);

    reg [2:0] state = u_if.IDLE;
    reg [3:0] counter = 0;
    reg [7:0] DATA_reg;

    always @(posedge u_if.clk or negedge u_if.reset) begin
        if (!u_if.reset) begin
            state <= u_if.IDLE;
            u_if.PAR_OUT_ex <= 0;
            u_if.RX_DONE_ex <= 0;
            counter <= 0;
            DATA_reg <= 0;
        end else begin
            case (state)
                u_if.IDLE: begin
                    DATA_reg <= 0;
                    counter <= 0;
                    u_if.PAR_OUT_ex <= 0;
                    u_if.RX_DONE_ex <= 0;
                    if (u_if.Busy) begin
                        state <= u_if.START;
                    end
                end
                u_if.START: begin
                    state <= u_if.DATA;
                    counter <= 0;
                end
                u_if.DATA: begin
                    DATA_reg[counter] <= u_if.TX_OUT;
                    counter <= counter + 1;
                    if (counter == 7) begin
                        state = (u_if.PAR_EN) ? u_if.PARITY : u_if.STOP;
                    end
                end
                u_if.PARITY: begin
                    u_if.PAR_OUT_ex <= u_if.TX_OUT;
                    state <= u_if.STOP;
                end
                u_if.STOP: begin
                    u_if.P_DATA_OUT_ex <= DATA_reg;
                    u_if.RX_DONE_ex <=1'b1 ;
                    state <= u_if.IDLE;
                end
            endcase
        end
    endmodule
```

interface :

```
interface uart_if (clk);
    input bit clk;
    logic reset;
    logic [7:0] P_DATA , P_DATA_OUT , P_DATA_OUT_ex;
    logic PAR_EN;
    logic PAR_TYP;
    logic PAR_OUT , PAR_OUT_ex;
    logic RX_DONE , RX_DONE_ex;
    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 TX ( input clk , reset , P_DATA , PAR_EN , PAR_TYP , DATA_VALID ,
                 output TX_OUT , Busy);

    modport RX (input TX_OUT , clk , reset , Busy , PAR_EN ,
                 output P_DATA_OUT , PAR_OUT , RX_DONE );

    modport GM_rx (input TX_OUT , clk , reset , Busy , PAR_EN ,
                   output P_DATA_OUT_ex , PAR_OUT_ex , RX_DONE_ex );

    modport GM_tx ( 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 , P_DATA_OUT , PAR_OUT
, RX_DONE , P_DATA_OUT_ex , PAR_OUT_ex , RX_DONE_ex);

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

endinterface
```

PKG :

```
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 ;
    logic PAR_OUT;
    logic RX_DONE;
    logic [7:0] P_DATA_OUT;

    rand int pattern_type;

    constraint reset_c {reset dist {0 := 3 , 1 := 97};}

    constraint PAR_TYP_c {PAR_TYP dist {0 := 50 , 1 := 50};}

    constraint PAR_EN_c {PAR_EN dist{0 := 75 , 1 := 25};}

    constraint DATA_VALID_c {DATA_VALID dist {0 := 8 , 1 := 92};} ;

    constraint P_DATA LSB_c {P_DATA[0] dist {0 := 20 , 1:=80};}

    constraint pattern_distribution { pattern_type dist {0 := 96 , 1 := 4 };}

    constraint P_DATA_c { if (pattern_type) P_DATA inside{8'hFF , 8'h00 , 8'hAA};}

covergroup cvr_gp ;
    RST_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_cross_TYPE : cross PAR_EN_cp , PAR_TYP_cp ;
    P_DATA_cross_DATA_VALID : cross P_DATA_cp , DATA_VALID_cp ;
    TX_OUT_cross_Busy : cross TX_OUT_cp , Busy_cp ;

    PAR_OUT_cp : coverpoint PAR_OUT {
        bins zero = {0} ;
        bins one = {1} ;
    }
    RX_DONE_cp : coverpoint RX_DONE {
        bins zero = {0} ;
        bins one = {1} ;
    }
    P_DATA_OUT_cp : coverpoint P_DATA_OUT {
        bins all_values  = {[0:255]};
    }
endgroup

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

endclass
endpackage
```

test bench :

```
import uart_pkg ::*;
module uart_tb (uart_if.TEST u_if) ;

uart_class uc = new () ;

int correct_counter = 0 ;
int error_counter = 0 ;

initial begin
    //UART_1
    assert_rst;
    //UART_2
    repeat(7864) begin
        assert (uc.randomize())
        u_if.reset = uc.reset ;
        u_if.DATA_VALID = uc.DATA_VALID;
        u_if.P_DATA = uc.P_DATA ;
        u_if.PAR_EN = uc.PAR_EN ;
        u_if.PAR_TYP = uc.PAR_TYP ;
        @(posedge u_if.clk);
        u_if.DATA_VALID = 1'b0;
        repeat (9) begin
            @(posedge u_if.clk) ;
            uc.TX_OUT = u_if.TX_OUT ;
            uc.Busy = u_if.Busy ;
            uc.RX_DONE = u_if.RX_DONE;
            uc.PAR_OUT = u_if.PAR_OUT;
            uc.P_DATA_OUT = u_if.P_DATA_OUT;
            @(posedge u_if.clk) ;
            check_result ;
        end
    end

    #2 $display ("NO. of Correct operations :      , No. of errors :      "
    ,correct_counter,error_counter);
    #2;
    $stop;
end

always @(posedge u_if.clk) begin
    uc.cvr_gp.sample();
end

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

task check_result ;
    if ((u_if.TX_OUT === u_if.TX_OUT_ex)&&(u_if.Busy === u_if.Busy_ex)
    &&(u_if.P_DATA_OUT === u_if.P_DATA_OUT_ex)&&(u_if.PAR_OUT === u_if.
    PAR_OUT_ex)&&(u_if.RX_DONE === u_if.RX_DONE_ex))
        correct_counter ++ ;
    else begin
        $display ("Error at time %t ", $time );
        error_counter++ ;
    end
endtask

endmodule
```

monitor :

```
uart_monitor.sv
1 module uart_monitor (uart_if.MONITOR u_if) ;
2   always @(posedge u_if.clk ) begin
3     $display("STIMULUS : Reset = 0b%0b , P_DATA = 0h%0h ,
4       PAR_EN = 0b%0b , PAR_TYP = 0b%0b ,
5       DATA_VALID = 0b%0b " , u_if.reset , u_if.P_DATA ,
6       u_if.PAR_EN , u_if.PAR_TYP ,
7       u_if.DATA_VALID);
8     $display("OUTPUT : TX_OUT = 0b%0b , TX_OUT_ex = 0b%0b ,
9       Busy = 0b%0b , Busy_ex = 0b%0b , P_DATA_OUT = 0h%0h , PARITY_OUT = 0b%0b , RX_DONE = 0b%0b" ,
10      u_if.TX_OUT,u_if.TX_OUT_ex,u_if.Busy,
11      u_if.Busy_ex,u_if.P_DATA_OUT,u_if.PAR_OUT,u_if.RX_DONE);
12   end
13 endmodule
```

top :

```
uart_top.sv
module uart_top ;
  bit clk ;
  initial begin
    forever #1 clk = ~clk ;
  end

  uart_if u_if (clk) ;
  uart_tb tb (u_if) ;
  uart_tx TX (u_if) ;
  uart_rx RX (u_if) ;
  uart_monitor MON (u_if) ;
  uart_golden_model_tx GM_tx (u_if) ;
  uart_golden_model_rx GM_rx (u_if) ;

  bind uart_tx uart_sva_tx svatx (u_if) ;
  bind uart_rx uart_sva_rx svarx (u_if) ;

endmodule
```

Do file :

```
# uart.do
1 vlib work
2 vlog -f src_files.list +cover -covercells
3 vsim -voptargs+=acc work_uart_top -cover
4 add wave *
5 add wave /uart_top/u_if/*
6 add wave /uart_top/RX/*
7 add wave /uart_top/TX/*
8 coverage exclude -du uart_tx -togglenode {state[3]}
9 coverage exclude -src uart_tx.sv -line 14 -code b
10 coverage save uart_cov.ucdb -onexit -du uart_tx
11 run -all
12 coverage exclude -cvxpath /uart_pkg/uart_class/cvr_gp/TX_OUT_cross_Busy/<zero,zero>
13 ##quit -sim
14 ##vcover report uart_cov.ucdb -details -annotate -all -output Coverage_rpt_UART_sv.txt
15
```

```
src_files.list
1 uart_if.sv
2 uart_tx.sv
3 uart_rx.sv
4 uart_golden_model_tx.sv
5 uart_golden_model_rx.sv
6 uart_pkg.sv
7 uart_tb.sv
8 uart_monitor.sv
9 uart_sva_tx.sv
10 uart_sva_rx.sv
11 uart_top.sv
12
```

SVA TX :

```

module uart_sva_tx (uart_if.TX u_if);

    always_comb begin
        if(!u_if.reset)
            assert_reset: assert final ((u_if.TX_OUT)&&(!u_if.Busy)&&(uart_tx.state==u_if.IDLE)&&(uart_tx.counter==0));
            cover_reset:cover final ((u_if.TX_OUT)&&(!u_if.Busy)&&(uart_tx.state==u_if.IDLE)&&(uart_tx.counter==0));
    end

    property IDLE_Behaviour1_p;
        @posedge u_if.clk disable iff(!u_if.reset)
            ((uart_tx.state==u_if.IDLE) && (~u_if.DATA_VALID) )|=>
                (u_if.TX_OUT == 1) && (!u_if.Busy) && (uart_tx.counter == 1'b0) ;
    endproperty

    property IDLE_Behaviour2_p;
        @posedge u_if.clk disable iff(!u_if.reset)
            ((uart_tx.state==u_if.IDLE) && (~u_if.DATA_VALID) )|=> (uart_tx.state==u_if.IDLE) ;
    endproperty

    property IDLE_Behaviour3_p;
        @posedge u_if.clk disable iff(!u_if.reset)
            ((uart_tx.state==u_if.IDLE)&&(u_if.DATA_VALID) && (u_if.PAR_TYP==0) |=>
                (uart_tx.PARITY_bit==$past(u_if.P_DATA)) && (u_if.Busy) && (uart_tx.DATA_reg==$past(u_if.P_DATA)) );
    endproperty

    property IDLE_Behaviour4_p;
        @posedge u_if.clk disable iff(!u_if.reset)
            (uart_tx.state==u_if.IDLE)&&(u_if.DATA_VALID) && (u_if.PAR_TYP==1) |=>
                (uart_tx.PARITY_bit==$past(u_if.P_DATA)) && (u_if.Busy) && (uart_tx.DATA_reg==$past(u_if.P_DATA)) ;
    endproperty

    property IDLE_Behaviour5_p;
        @posedge u_if.clk disable iff(!u_if.reset)
            (uart_tx.state==u_if.IDLE)&&(u_if.DATA_VALID) |=> (uart_tx.state==u_if.START);
    endproperty

    property START_Behaviour_p;
        @posedge u_if.clk disable iff(!u_if.reset)
            (uart_tx.state==u_if.START) |=>
                (u_if.TX_OUT==0) && (uart_tx.state==u_if.DATA) && (uart_tx.counter==0);
    endproperty

    property DATA_Behaviour_p;
        @posedge u_if.clk disable iff(!u_if.reset)
            ((uart_tx.state==u_if.DATA) && (uart_tx.counter != 7) )|=>
                ((u_if.TX_OUT)==uart_tx.DATA_reg[$past(uart_tx.counter)]) && (uart_tx.counter==$past(uart_tx.counter)+1);
    endproperty

    property DATA_Behaviour1_p;
        @posedge u_if.clk disable iff(!u_if.reset)
            ((uart_tx.state==u_if.DATA) && (uart_tx.counter != 7) )|=> (uart_tx.state==u_if.DATA) ;
    endproperty

    property DATA_Behaviour2_p;
        @posedge u_if.clk disable iff(!u_if.reset)
            (uart_tx.state==u_if.DATA)&&(uart_tx.counter==7)&&(u_if.PAR_EN==1) |=>
                (uart_tx.state==u_if.PARITY);
    endproperty

    property DATA_Behaviour3_p;
        @posedge u_if.clk disable iff(!u_if.reset)
            (uart_tx.state==u_if.DATA)&&(uart_tx.counter==7)&&(u_if.PAR_EN==0) |=>
                (uart_tx.state==u_if.STOP);
    endproperty

    property PARITY_Behaviour_p;
        @posedge u_if.clk disable iff(!u_if.reset)
            (uart_tx.state==u_if.PARITY) |=>
                (u_if.TX_OUT==$past(uart_tx.PARITY_bit)) &&(uart_tx.state==u_if.STOP);
    endproperty

    property STOP_Behaviour_p;
        @posedge u_if.clk disable iff(!u_if.reset)
            (uart_tx.state==u_if.STOP) |=>
                (u_if.TX_OUT==1) &&(uart_tx.state==u_if.IDLE);
    endproperty

    // IDLE State Properties
    assert property (IDLE_Behaviour1_p);
    cover property (IDLE_Behaviour1_p);

    assert property (IDLE_Behaviour2_p);
    cover property (IDLE_Behaviour2_p);

    assert property (IDLE_Behaviour3_p);
    cover property (IDLE_Behaviour3_p);

    assert property (IDLE_Behaviour4_p);
    cover property (IDLE_Behaviour4_p);

    assert property (IDLE_Behaviour5_p);
    cover property (IDLE_Behaviour5_p);

    // START State Property
    assert property (START_Behaviour_p);
    cover property (START_Behaviour_p);

    // DATA State Properties
    assert property (DATA_Behaviour_p);
    cover property (DATA_Behaviour_p);

    assert property (DATA_Behaviour1_p);
    cover property (DATA_Behaviour1_p);

    assert property (DATA_Behaviour2_p);
    cover property (DATA_Behaviour2_p);

    assert property (DATA_Behaviour3_p);
    cover property (DATA_Behaviour3_p);

    // PARITY State Property
    assert property (PARITY_Behaviour_p);
    cover property (PARITY_Behaviour_p);

    // STOP State Property
    assert property (STOP_Behaviour_p);
    cover property (STOP_Behaviour_p);

endmodule

```

SVA RX :

```

module uart_sva_rx (uart_if.RX u_if);

    always_comb begin
        if(!u_if.reset)
            assert_reset: assert final ((uart_rx.DATA_reg==0)&&(!u_if.RX_DONE)&&(u_if.PAR_OUT)&&(uart_rx.cs==u_if.IDLE)&&(uart_rx.counter==0));
            cover_reset:cover final ((uart_rx.DATA_reg==0)&&(!u_if.RX_DONE)&&(uart_rx.cs==u_if.IDLE)&&(uart_rx.counter==0));
    end

    property IDLE_Behaviour1_p;
        @posedge u_if.clk disable iff(!u_if.reset)
            ((uart_rx.cs==u_if.IDLE) && (u_if.Busy) )|=>
                (uart_rx.ns == u_if.START) [=1];
    endproperty

    property IDLE_Behaviour2_p;
        @posedge u_if.clk disable iff(!u_if.reset)
            ((uart_rx.cs==u_if.IDLE))|=>
                (uart_rx.DATA_reg==0);
    endproperty

    property IDLE_Behaviour3_p;
        @posedge u_if.clk disable iff(!u_if.reset)
            ((uart_rx.cs==u_if.IDLE))|=>
                (uart_rx.counter==0);
    endproperty

    property IDLE_Behaviour4_p;
        @posedge u_if.clk disable iff(!u_if.reset)
            ((uart_rx.cs==u_if.IDLE))|=>
                (u_if.PAR_OUT==0);
    endproperty

    property IDLE_Behaviour5_p;
        @posedge u_if.clk disable iff(!u_if.reset)
            ((uart_rx.cs==u_if.IDLE))|=>
                (u_if.RX_DONE==0);
    endproperty

    property START_Behaviour_p;
        @posedge u_if.clk disable iff(!u_if.reset)
            (uart_rx.cs==u_if.START) |=>
                (uart_rx.ns==u_if.DATA) && (uart_rx.counter==0);
    endproperty

    property DATA_Behaviour_p;
        @posedge u_if.clk disable iff(!u_if.reset)
            ((uart_rx.cs==u_if.DATA)) |=>
                (uart_rx.DATA_reg[$past(uart_rx.counter)] == $past(u_if.TX_OUT)) && (uart_rx.counter==$past(uart_rx.counter)+1);
    endproperty

    property DATA_Behaviour1_p;
        @posedge u_if.clk disable iff(!u_if.reset)
            (uart_rx.cs==u_if.DATA)&&(uart_rx.counter==7)&&(u_if.PAR_EN==1) |=>
                (uart_rx.ns==u_if.PARITY)[=1];
    endproperty

    property DATA_Behaviour2_p;
        @posedge u_if.clk disable iff(!u_if.reset)
            (uart_rx.cs==u_if.DATA)&&(uart_rx.counter==7)&&(u_if.PAR_EN==0) |=>
                (uart_rx.ns==u_if.STOP)[=1];
    endproperty

    property PARITY_Behaviour1_p;
        @posedge u_if.clk disable iff(!u_if.reset)
            (uart_rx.cs==u_if.PARITY) |=>
                (u_if.PAR_OUT==$past(u_if.TX_OUT));
    endproperty

    property PARITY_Behaviour2_p;
        @posedge u_if.clk disable iff(!u_if.reset)
            (uart_rx.cs==u_if.PARITY) |=>
                (uart_rx.ns==u_if.STOP)[=1];
    endproperty

    property STOP_Behaviour_p;
        @posedge u_if.clk disable iff(!u_if.reset)
            (uart_rx.cs==u_if.STOP) |=>
                (u_if.P_DATA_OUT==$past(uart_rx.DATA_reg)) && (u_if.RX_DONE) && (uart_rx.ns==u_if.IDLE);
    endproperty

    // IDLE State Properties
    assert property (IDLE_Behaviour1_p);
    cover property (IDLE_Behaviour1_p);

    assert property (IDLE_Behaviour2_p);
    cover property (IDLE_Behaviour2_p);

    assert property (IDLE_Behaviour3_p);
    cover property (IDLE_Behaviour3_p);

    assert property (IDLE_Behaviour4_p);
    cover property (IDLE_Behaviour4_p);

    assert property (IDLE_Behaviour5_p);
    cover property (IDLE_Behaviour5_p);

    // START State Property
    assert property (START_Behaviour_p);
    cover property (START_Behaviour_p);

    // DATA State Properties
    assert property (DATA_Behaviour_p);
    cover property (DATA_Behaviour_p);

    assert property (DATA_Behaviour1_p);
    cover property (DATA_Behaviour1_p);

    assert property (DATA_Behaviour2_p);
    cover property (DATA_Behaviour2_p);

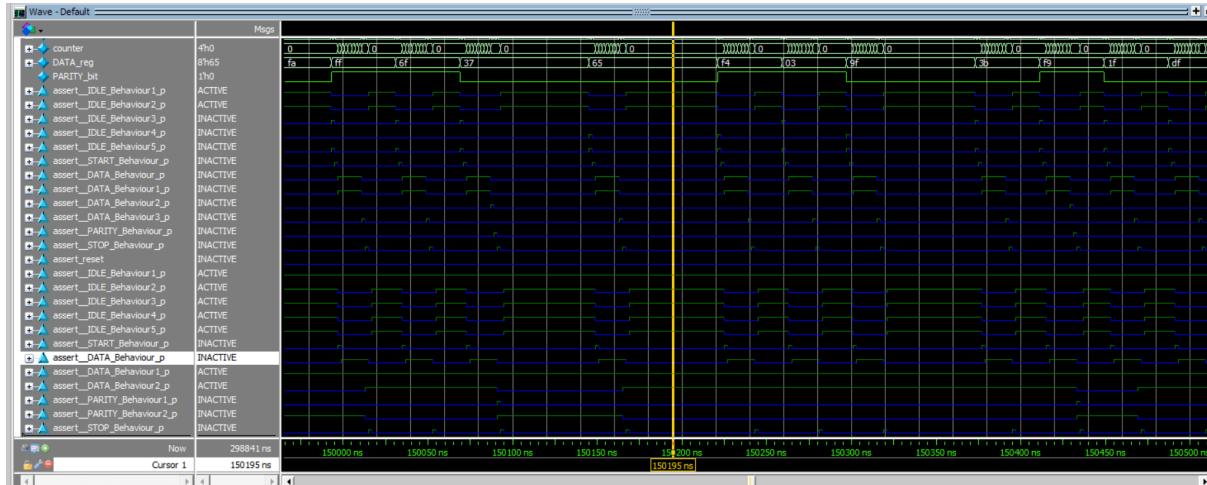
    // PARITY State Property
    assert property (PARITY_Behaviour1_p);
    cover property (PARITY_Behaviour1_p);

    assert property (PARITY_Behaviour2_p);
    cover property (PARITY_Behaviour2_p);

    // STOP State Property
    assert property (STOP_Behaviour_p);
    cover property (STOP_Behaviour_p);

endmodule

```



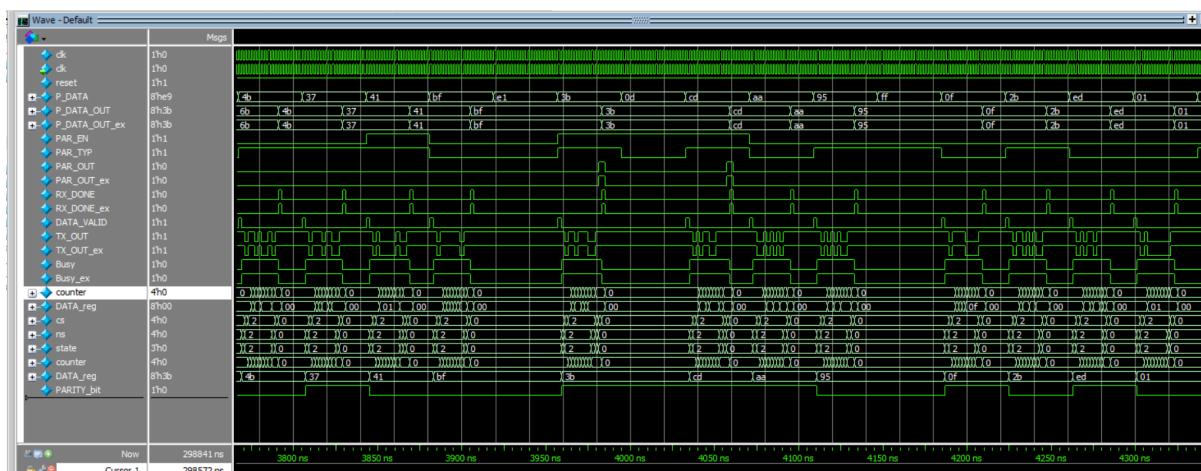
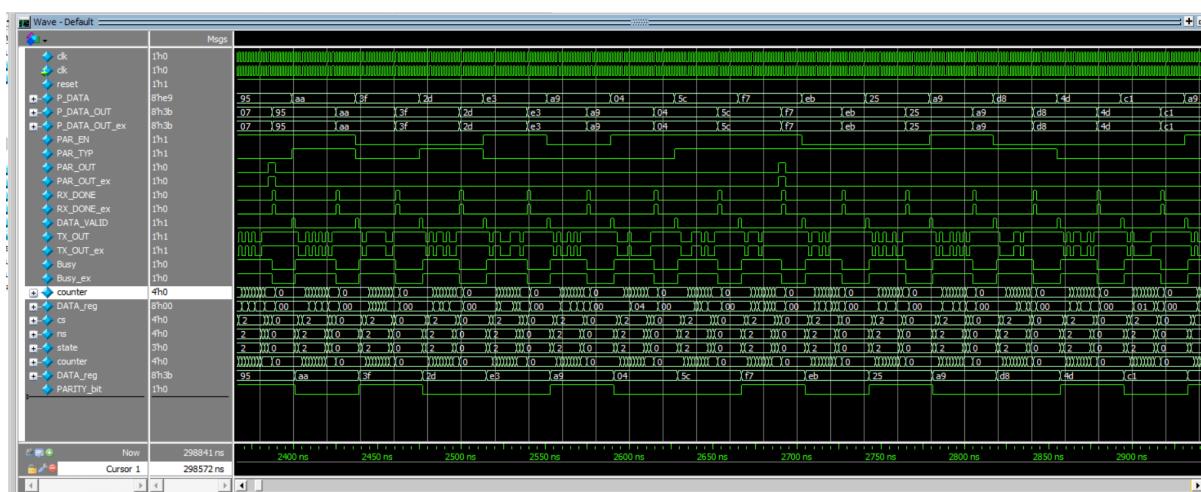
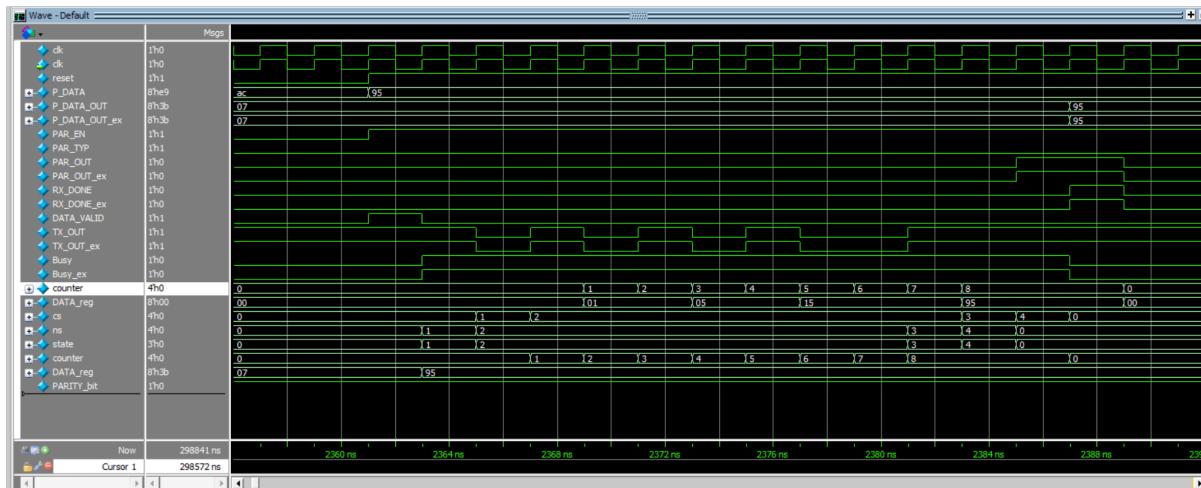
Assertions

Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Count	Memory	Peak Memory	Peak Memory Time	Cumulative Threads	ATV	Assertion Expression	Included
uart_top_tb/ublk#193570514#13/mmed_14	Immediate	SVA	on	0	1	-	-	-	-	0 off	assert (randomize(...))	✓	
uart_top/TX/svab/assert_reset	Immediate	SVA	on	0	1	-	-	-	-	0 off	assert(u_if.TX_OUT<=u_if.Busy&u_if	✓	
uart_top/TX/svab/assert_IDLE_Behaviour1_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert! (@posedge u_if.clk) disable...	✓	
uart_top/TX/svab/assert_IDLE_Behaviour2_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert! (@posedge u_if.clk) disable...	✓	
uart_top/TX/svab/assert_IDLE_Behaviour3_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert! (@posedge u_if.clk) disable...	✓	
uart_top/TX/svab/assert_IDLE_Behaviour4_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert! (@posedge u_if.clk) disable...	✓	
uart_top/TX/svab/assert_IDLE_Behaviour5_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert! (@posedge u_if.clk) disable...	✓	
uart_top/TX/svab/assert_START_Behaviour_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert! (@posedge u_if.clk) disable...	✓	
uart_top/TX/svab/assert_DATA_Behaviour_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert! (@posedge u_if.clk) disable...	✓	
uart_top/TX/svab/assert_DATA_Behaviour1_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert! (@posedge u_if.clk) disable...	✓	
uart_top/TX/svab/assert_DATA_Behaviour2_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert! (@posedge u_if.clk) disable...	✓	
uart_top/TX/svab/assert_DATA_Behaviour3_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert! (@posedge u_if.clk) disable...	✓	
uart_top/TX/svab/assert_STOP_Behaviour_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert! (@posedge u_if.clk) disable...	✓	
uart_top/RX/svab/assert_RESET	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert(u_if.RX_IN==1) & !DATA_in & !PARITY_in & !STOP_in	✓	
uart_top/RX/svab/assert_IDLE_Behaviour1_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert! (@posedge u_if.clk) disable...	✓	
uart_top/RX/svab/assert_IDLE_Behaviour2_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert! (@posedge u_if.clk) disable...	✓	
uart_top/RX/svab/assert_IDLE_Behaviour3_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert! (@posedge u_if.clk) disable...	✓	
uart_top/RX/svab/assert_IDLE_Behaviour4_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert! (@posedge u_if.clk) disable...	✓	
uart_top/RX/svab/assert_IDLE_Behaviour5_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert! (@posedge u_if.clk) disable...	✓	
uart_top/RX/svab/assert_START_Behaviour_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert! (@posedge u_if.clk) disable...	✓	
uart_top/RX/svab/assert_DATA_Behaviour_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert! (@posedge u_if.clk) disable...	✓	
uart_top/RX/svab/assert_DATA_Behaviour1_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert! (@posedge u_if.clk) disable...	✓	
uart_top/RX/svab/assert_DATA_Behaviour2_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert! (@posedge u_if.clk) disable...	✓	
uart_top/RX/svab/assert_PARITY_Behaviour1_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert! (@posedge u_if.clk) disable...	✓	
uart_top/RX/svab/assert_PARITY_Behaviour2_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert! (@posedge u_if.clk) disable...	✓	
uart_top/RX/svab/assert_PARITY_Behaviour3_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert! (@posedge u_if.clk) disable...	✓	
uart_top/RX/svab/assert_PARITY_Behaviour4_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert! (@posedge u_if.clk) disable...	✓	
uart_top/RX/svab/assert_PARITY_Behaviour5_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert! (@posedge u_if.clk) disable...	✓	
uart_top/RX/svab/assert_STOP_Behaviour_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ns	0 off	assert! (@posedge u_if.clk) disable...	✓	

Cover Directives

Name	Language	Enabled	Log	Count	AtLeast	Limit	Weight	Chmt %	Chmt graph	Included	Memory	Peak Memory	Peak Memory Time	Cumulative Threads
uart_top/TX/svab/cover_STOP_Behaviour_p	SVA	Off	7013	1	Unlimited	1	100%	✓	✓	0	0	0	0 ns	0
uart_top/TX/svab/cover_PARITY_Behaviour_p	SVA	✓	Off	1702	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0
uart_top/TX/svab/cover_DATA_Behaviour3_p	SVA	✓	Off	5311	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0
uart_top/TX/svab/cover_DATA_Behaviour2_p	SVA	✓	Off	1702	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0
uart_top/TX/svab/cover_DATA_Behaviour1_p	SVA	✓	Off	4991	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0
uart_top/TX/svab/cover_DATA_Behaviour_p	SVA	✓	Off	4991	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0
uart_top/TX/svab/cover_START_Behaviour_p	SVA	✓	Off	7013	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0
uart_top/RX/svab/cover_STOP_Behaviour_p	SVA	✓	Off	1702	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0
uart_top/RX/svab/cover_PARITY_Behaviour2_p	SVA	✓	Off	1654	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0
uart_top/RX/svab/cover_PARITY_Behaviour1_p	SVA	✓	Off	1702	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0
uart_top/RX/svab/cover_DATA_Behaviour2_p	SVA	✓	Off	5131	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0
uart_top/RX/svab/cover_DATA_Behaviour1_p	SVA	✓	Off	1489	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0
uart_top/RX/svab/cover_DATA_Behaviour_p	SVA	✓	Off	55104	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0
uart_top/RX/svab/cover_IDLE_Behaviour1_p	SVA	✓	Off	7013	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0
uart_top/RX/svab/cover_IDLE_Behaviour2_p	SVA	✓	Off	1654	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0
uart_top/RX/svab/cover_IDLE_Behaviour3_p	SVA	✓	Off	1702	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0
uart_top/RX/svab/cover_IDLE_Behaviour4_p	SVA	✓	Off	3496	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0
uart_top/RX/svab/cover_IDLE_Behaviour5_p	SVA	✓	Off	3517	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0
uart_top/RX/svab/cover_IDLE_Behaviour2_p	SVA	✓	Off	65783	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0
uart_top/RX/svab/cover_IDLE_Behaviour1_p	SVA	✓	Off	65783	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0
uart_top/TX/svab/cover_RESET	SVA	✓	Off	7471	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0
uart_top/RX/svab/cover_STOP_Behaviour_p	SVA	✓	Off	7013	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0
uart_top/RX/svab/cover_PARITY_Behaviour2_p	SVA	✓	Off	1654	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0
uart_top/RX/svab/cover_PARITY_Behaviour1_p	SVA	✓	Off	1702	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0
uart_top/RX/svab/cover_DATA_Behaviour1_p	SVA	✓	Off	5131	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0
uart_top/RX/svab/cover_DATA_Behaviour_p	SVA	✓	Off	55104	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0
uart_top/RX/svab/cover_START_Behaviour_p	SVA	✓	Off	7013	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0
uart_top/RX/svab/cover_IDLE_Behaviour5_p	SVA	✓	Off	72797	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0
uart_top/RX/svab/cover_IDLE_Behaviour4_p	SVA	✓	Off	72797	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0
uart_top/RX/svab/cover_IDLE_Behaviour3_p	SVA	✓	Off	72797	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0
uart_top/RX/svab/cover_IDLE_Behaviour2_p	SVA	✓	Off	72797	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0
uart_top/RX/svab/cover_IDLE_Behaviour1_p	SVA	✓	Off	6785	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0
uart_top/RX/svab/cover_RESET	SVA	✓	Off	7471	1	Unlimited	1	100%	✓	✓	0	0	0 ns	0

Simulation :



```

#
#           Busy = 0b0 , Busy_ex = 0b0 , P_DATA_OUT = 0hlf , PARITY_OUT = 0b0 , RX_DONE = 0b0
# STIMULUS : Reset = 0b1 , P_DATA = 0hlf ,
#
#           PAR_EN = 0b0 , PAR_TYP = 0b0 ,
#
#           DATA_VALID = 0b0
# OUTPUT : TX_OUT = 0b1 , TX_OUT_ex = 0b1 ,
#
#           Busy = 0b0 , Busy_ex = 0b0 , P_DATA_OUT = 0hlf , PARITY_OUT = 0b0 , RX_DONE = 0b0
# STIMULUS : Reset = 0b1 , P_DATA = 0h89 ,
#
#           PAR_EN = 0b0 , PAR_TYP = 0b0 ,
#
#           DATA_VALID = 0b1
# OUTPUT : TX_OUT = 0b1 , TX_OUT_ex = 0b1 ,
#
#           Busy = 0b0 , Busy_ex = 0b0 , P_DATA_OUT = 0hlf , PARITY_OUT = 0b0 , RX_DONE = 0b0
# STIMULUS : Reset = 0b1 , P_DATA = 0h89 ,
#
#           PAR_EN = 0b0 , PAR_TYP = 0b0 ,
#
#           DATA_VALID = 0b0
# OUTPUT : TX_OUT = 0b1 , TX_OUT_ex = 0b1 ,
#
#           Busy = 0b1 , Busy_ex = 0b1 , P_DATA_OUT = 0hlf , PARITY_OUT = 0b0 , RX_DONE = 0b0
# STIMULUS : Reset = 0b1 , P_DATA = 0h89 ,
#
#           PAR_EN = 0b0 , PAR_TYP = 0b0 ,
#
# ** Note: (vsim-3813) Design is being optimized due to module recompilation...
# Loading sv_std.std
# Loading work uart_top(fast)
# Loading work uart_if(fast_1)
# Loading work uart_pkg(fast)
# Loading work uart_tb_sv_unit(fast)
# Loading work uart_tb(fast)
# Loading work uart_tx(fast)
# Loading work uart_sva_tx(fast)
# Loading work uart_rx(fast)
# Loading work uart_sva_rx(fast)
# Loading work uart_golden_model_tx(fast)
# Loading work uart_golden_model_rx(fast)
# coverage exclude -du uart_tx -togglenode {state[3]}
# ** UI-Msg: (vsim-4036) The 'coverage exclude' command had no effect on some/all objects because
# no matching coverage data was found.
#
# NO. of Correct operations : 70776 , No. of errors : 0
# ** Note: $stop    : uart_tb.sv(37)
#   Time: 298841 ns  Iteration: 0  Instance: /uart_top/tb
# Break in Module uart_tb at uart_tb.sv line 37

```

Code Coverage :

```
Statement Coverage: /4/1 covered
  Enabled Coverage      Bins    Hits    Misses  Coverage
  -----      -----
  Statements          1       1        0   100.00%
=====
Statement Details=====
Statement Coverage for instance /\uart_top#TX /svatx --
  Line      Item      Count      Source
  ----      ---      -----
  File uart_sva_tx.sv
    1                      module uart_sva_tx (uart_if.TX u_if);
    2
    3           1           86317      always_comb begin
  ===
  == Instance: /\uart_top#TX
  == Design Unit: work_uart_tx
  =====
Branch Coverage:
  Enabled Coverage      Bins    Hits    Misses  Coverage
  -----      -----
  Branches            10       10        0   100.00%
```

```
Condition Coverage:
  Enabled Coverage      Bins    Covered    Misses  Coverage
  -----      -----
  Conditions          1         1        0   100.00%
=====
Condition Details=====
Condition Coverage for instance /\uart_top#TX --
  File uart_tx.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:		Bins	Hits	Misses	Coverage
Enabled	Coverage	-----	-----	-----	-----
Statements		22	22	0	100.00%

Toggle Coverage:		Bins	Hits	Misses	Coverage
Enabled	Coverage	-----	-----	-----	-----
Toggles		32	32	0	100.00%

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

Toggle Coverage for instance /\uart_top#TX --

Node	1H->0L	0L->1H	"Coverage"
DATA_reg[7-0]	1	1	100.00
PARITY_bit	1	1	100.00
counter[3-0]	1	1	100.00
state[2-0]	1	1	100.00

Total Node Count = 16

Toggled Node Count = 16

Untoggled Node Count = 0

Toggle Coverage = 100.00% (32 of 32 bins)

Covergroup :

Name	Class Type	Coverage	Goal	% of Goal	Status	Included
/uart_pkg/uart_class		100.00%				
TYPE cvr_gp		100.00%	100	100.00...		✓
CVP cvr_gp::RST_cp		100.00%	100	100.00...		✓
B bin zero		4560	1	100.00...		✓
B bin one		144857	1	100.00...		✓
CVP cvr_gp::PAR_TYP_cp		100.00%	100	100.00...		✓
B bin zero		75126	1	100.00...		✓
B bin one		74291	1	100.00...		✓
CVP cvr_gp::PAR_EN_cp		100.00%	100	100.00...		✓
B bin zero		112823	1	100.00...		✓
B bin one		36594	1	100.00...		✓
CVP cvr_gp::P_DATA_cp		100.00%	100	100.00...		✓
B bin all_values		149417	1	100.00...		✓
CVP cvr_gp::DATA_VALID_cp		100.00%	100	100.00...		✓
B bin zero		11989	1	100.00...		✓
B bin one		137428	1	100.00...		✓
CVP cvr_gp::TX_OUT_cp		100.00%	100	100.00...		✓
B bin zero		24854	1	100.00...		✓
B bin one		124561	1	100.00...		✓
CVP cvr_gp::Busy_cp		100.00%	100	100.00...		✓
B bin zero		65259	1	100.00...		✓
B bin one		84156	1	100.00...		✓
CVP cvr_gp::PAR_OUT_cp		100.00%	100	100.00...		✓
B bin zero		147709	1	100.00...		✓
B bin one		1706	1	100.00...		✓
CVP cvr_gp::RX_DONE_cp		100.00%	100	100.00...		✓
B bin zero		146011	1	100.00...		✓
B bin one		3404	1	100.00...		✓
CVP cvr_gp::P_DATA_OUT_cp		100.00%	100	100.00...		✓
B bin all_values		149403	1	100.00...		✓
CROSS cvr_gp::PAR_EN_cross_TYPE		100.00%	100	100.00...		✓
B bin <one,one>		17974	1	100.00...		✓
B bin <zero,one>		56317	1	100.00...		✓
B bin <one,zero>		18620	1	100.00...		✓
B bin <zero,zero>		56506	1	100.00...		✓
CROSS cvr_gp::P_DATA_cross_DATA_VALID		100.00%	100	100.00...		✓
B bin <all_values,one>		137428	1	100.00...		✓
B bin <all_values,zero>		11989	1	100.00...		✓
CROSS cvr_gp::TX_OUT_cross_Busy		100.00%	100	100.00...		✓
B bin <one,one>		59302	1	100.00...		✓
B bin <zero,one>		24854	1	100.00...		✓
B bin <one,zero>		65259	1	100.00...		✓