UART Design and Implementation using Verilog HDL

Modular Design, Verification, and Integration with FIFOs

Prepared by: Mohamed Essam

Date: August 2025

Contents

1. Introduction

In this project, a Universal Asynchronous Receiver/Transmitter (UART) was designed and implemented using Verilog HDL. The design follows a modular approach, where each block (Transmitter, Receiver, Baud Rate Generator, and FIFO Buffers) was implemented, tested individually with testbenches, and then integrated into a top-level UART system.

2. Design Methodology

The UART design process was divided into smaller modules. Each module was designed, verified using testbenches, and then connected together to form the complete UART system.

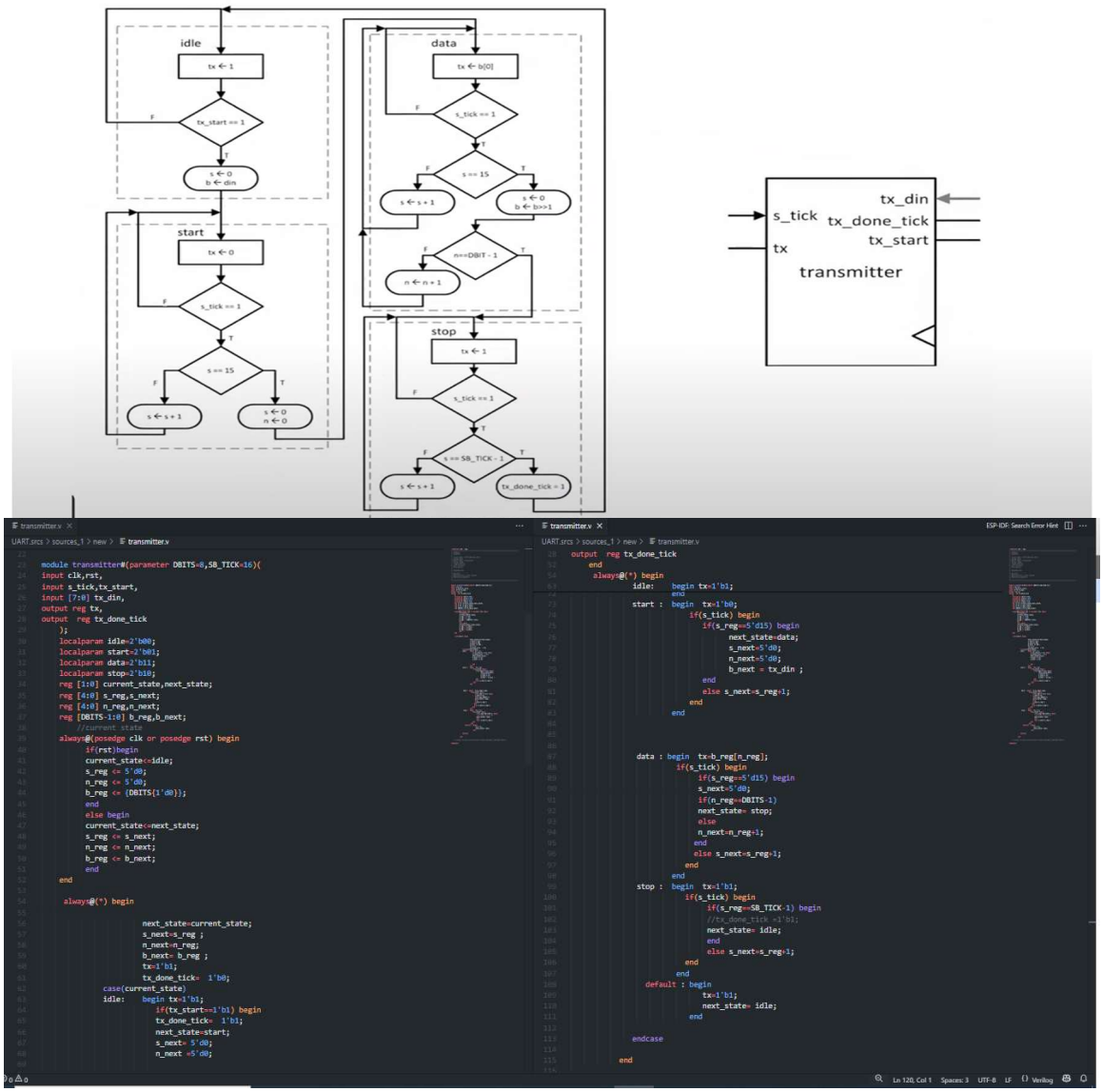
2.1 Baud Rate Generator

Function: Generates sampling ticks (s_tick) for synchronization.
Inputs: clk, rst, FINAL_VALUE
Outputs: done (s_tick)

```
module baud_rate_generator #(parameter BITS=10)(
input clk ,rst,en,
input [BITS-1:0]FINAL_VALUE,
output done
    );

    reg [BITS-1:0] Q_reg,Q_next;
    always @(posedge clk or posedge rst) begin
       if(rst)
       Q_reg<={BITS{1'b0}};
       else if(en)
        Q_reg<=Q_next;
     end
     always @(*) begin
         Q_next=done?{BITS{1'b0}}: Q_reg+1;
     end
    assign done =( Q_reg==FINAL_VALUE);

endmodule
```

2.2 Transmitter

Function: Serializes parallel data and transmits with start and stop bits.
Inputs: clk, rst, tx_din, tx_start, s_tick
Outputs: tx, tx_done_tick

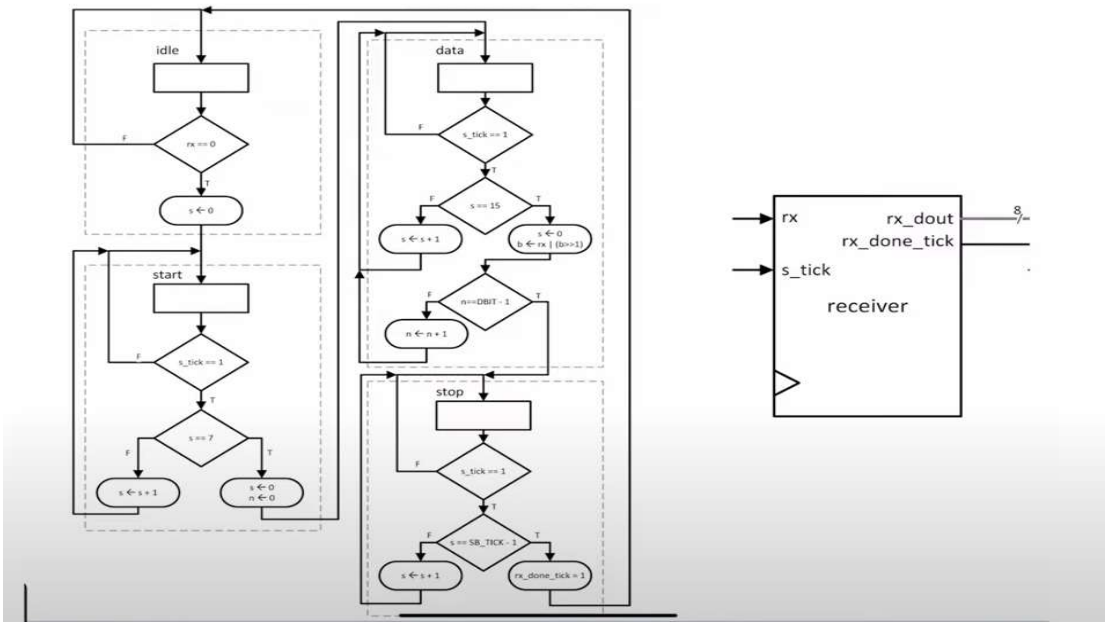


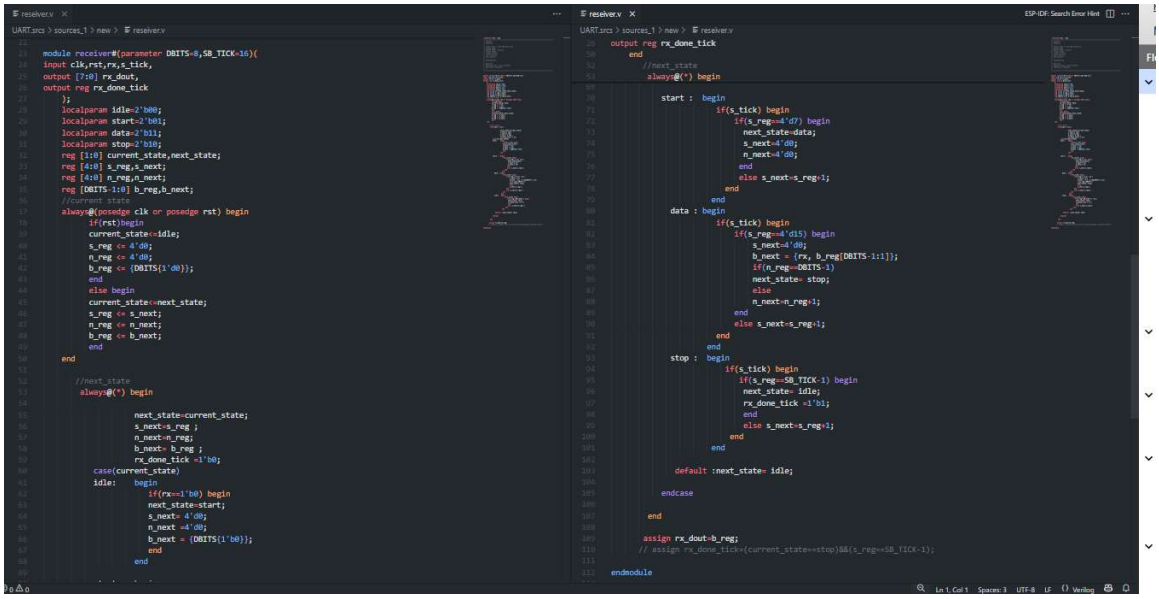
2.3 Receiver

Function: Deserializes incoming serial data into parallel format.

Inputs: clk, rst, rx, s_tick

Outputs: rx_dout, rx_done_tick



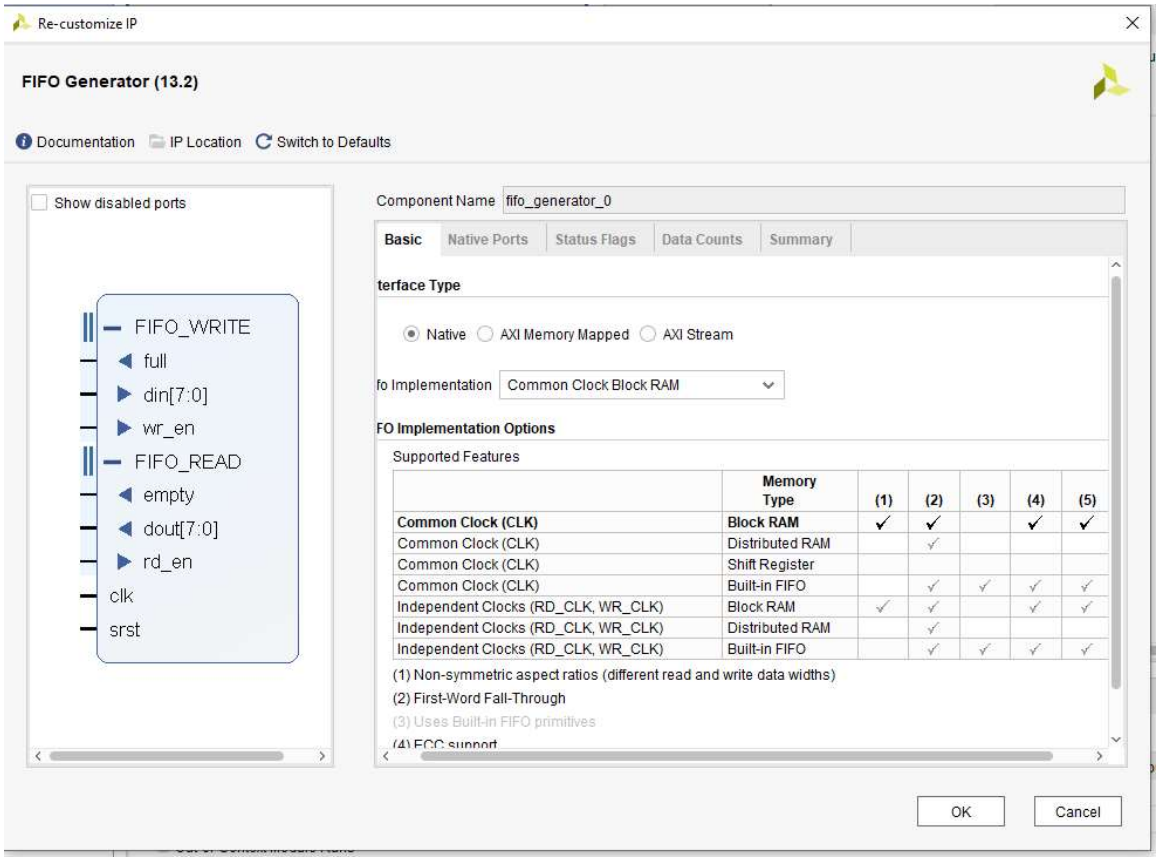


```
module receiver#(parameter DBITS=8,SB_TICK=16)(
input clk,rst,rx,s_tick,
output [7:0] rx_dout,
output reg rx_done_tick
    );
    localparam idle=2'b00;
    localparam start=2'b01;
    localparam data=2'b11;
    localparam stop=2'b10;
    reg [1:0] current_state,next_state;
    reg [4:0] s_reg,s_next;
    reg [4:0] n_reg,n_next;
    reg [DBITS-1:0] b_reg,b_next;
    //current state
    always@(posedge clk or posedge rst) begin
        if(rst)begin
        current_state<=idle;
        s_reg <= 4'd0;
        n_reg <= 4'd0;
        b_reg <= {DBITS{1'd0}};
        end
        else begin
        current_state<=next_state;
        s_reg <= s_next;
        n_reg <= n_next;
        b_reg <= b_next;
        end
    end

      //next_state
      always@(*) begin

                next_state=current_state;
                s_next=s_reg ;
                n_next=n_reg;
                b_next= b_reg ;
                rx_done_tick =1'b0;
        case(current_state)
        idle:   begin
                    if(rx==1'b0) begin
                    next_state=start;
                    s_next= 4'd0;
                    n_next =4'd0;
                    b_next = {DBITS{1'b0}};
                    end
                end
```

```
output reg rx_done_tick
    end
      //next_state
      always@(*) begin

        start :  begin
                    if(s_tick) begin
                        if(s_reg==4'd7) begin
                         next_state=data;
                         s_next=4'd0;
                         n_next=4'd0;
                        end
                        else s_next=s_reg+1;
                    end
                end
        data : begin
                    if(s_tick) begin
                        if(s_reg==4'd15) begin
                            s_next=4'd0;
                            b_next = {rx, b_reg[DBITS-1:1]};
                            if(n_reg==DBITS-1)
                            next_state= stop;
                            else
                            n_next=n_reg+1;
                        end
                        else s_next=s_reg+1;
                    end
                end
        stop :  begin
                    if(s_tick) begin
                        if(s_reg==SB_TICK-1) begin
                         next_state= idle;
                         rx_done_tick =1'b1;
                         end
                        else s_next=s_reg+1;
                    end
                end

        default :next_state= idle;

      endcase

    end

    assign rx_dout=b_reg;
   // assign rx_done_tick={current_state==stop}&&(s_reg==SB_TICK-1);

endmodule
```

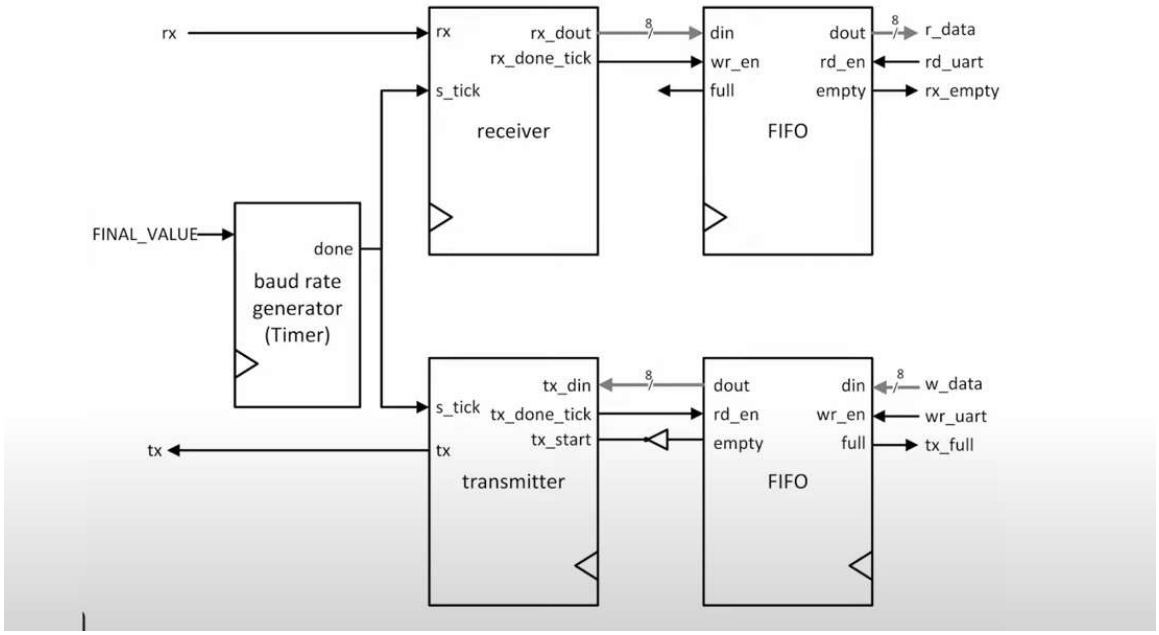
2.4 FIFO Buffers

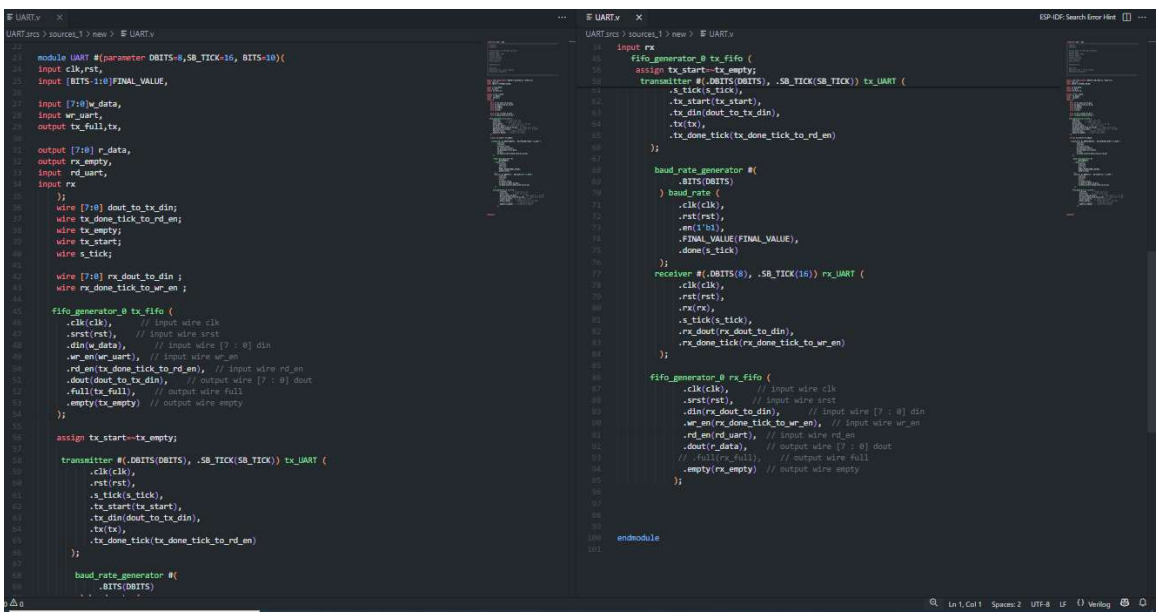
Function: Buffering mechanism for transmit and receive data.
Inputs: din, wr_en, rd_en
Outputs: dout, full, empty



3. System Integration

The transmitter, receiver, baud rate generator, and FIFO buffers were integrated into a top-level UART design. The final UART module allows bidirectional data transfer with buffering and configurable baud rate.

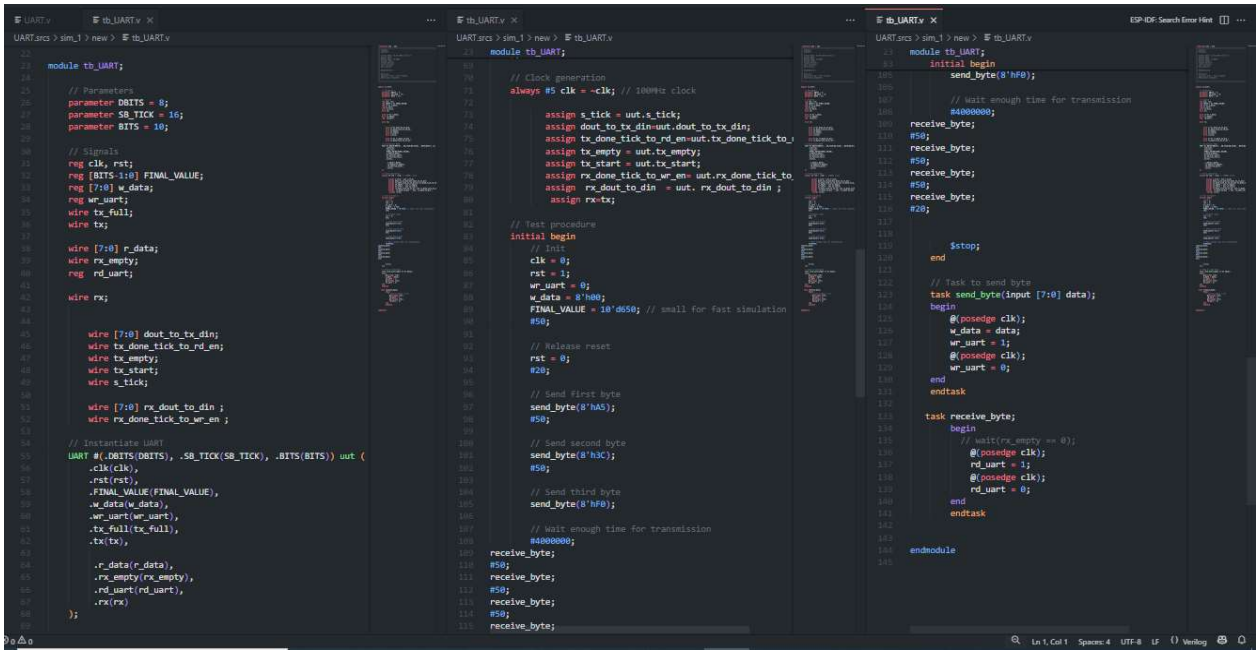




4. Verification

Each module was verified using individual testbenches. Simulation results confirmed correct serialization, deserialization, and data buffering. Waveforms and FSM diagrams were analyzed for validation.

4.1 Testbench Setup

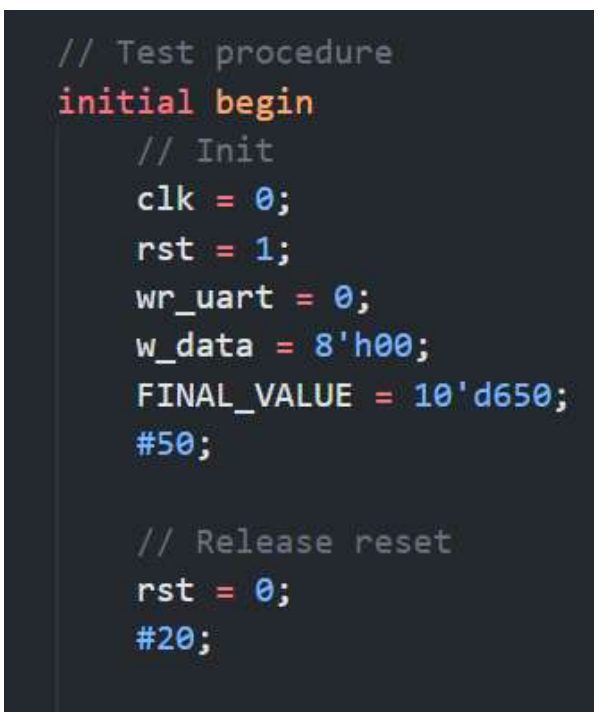


```
module tb_UART;

    // Parameters
    parameter DBITS = 8;
    parameter SB_TICK = 16;
    parameter BITS = 10;

    // Signals
    reg clk, rst;
    reg [BITS-1:0] FINAL_VALUE;
    reg [7:0] w_data;
    reg wr_uart;
    wire tx_full;
    wire tx;

    wire [7:0] r_data;
    wire rx_empty;
    reg  rd_uart;

    wire rx;


        wire [7:0] dout_to_tx_din;
        wire tx_done_tick_to_rd_en;
        wire tx_empty;
        wire tx_start;
        wire s_tick;

        wire [7:0] rx_dout_to_din ;
        wire rx_done_tick_to_wr_en ;

    // Instantiate UART
    UART #(.DBITS(DBITS), .SB_TICK(SB_TICK), .BITS(BITS)) uut (
        .clk(clk),
        .rst(rst),
        .FINAL_VALUE(FINAL_VALUE),
        .w_data(w_data),
        .wr_uart(wr_uart),
        .tx_full(tx_full),
        .tx(tx),

         .r_data(r_data),
         .rx_empty(rx_empty),
         .rd_uart(rd_uart),
         .rx(rx)
    );
```

```
module tb_UART;

    // Clock generation
    always #5 clk = ~clk; // 100MHz clock

          assign s_tick = uut.s_tick;
          assign dout_to_tx_din=uut.dout_to_tx_din;
          assign tx_done_tick_to_rd_en=uut.tx_done_tick_to_
          assign tx_empty = uut.tx_empty;
          assign tx_start = uut.tx_start;
          assign rx_done_tick_to_wr_en= uut.rx_done_tick_to
          assign  rx_dout_to_din  = uut. rx_dout_to_din ;
           assign rx=tx;

    // Test procedure
    initial begin
        // Init
        clk = 0;
        rst = 1;
        wr_uart = 0;
        w_data = 8'h00;
        FINAL_VALUE = 10'd650; // small for fast simulation
        #50;

        // Release reset
        rst = 0;
        #20;

        // Send first byte
        send_byte(8'hA5);
        #50;

        // Send second byte
        send_byte(8'h3C);
        #50;

        // Send third byte
        send_byte(8'hF0);

        // Wait enough time for transmission
        #4000000;
receive_byte;
#50;
receive_byte;
#50;
receive_byte;
#50;
receive_byte;
```

```
module tb_UART;
    initial begin
        send_byte(8'hF0);

        // Wait enough time for transmission
        #4000000;
receive_byte;
#50;
receive_byte;
#50;
receive_byte;
#50;
receive_byte;
#20;


        $stop;
    end

    // Task to send byte
    task send_byte(input [7:0] data);
    begin
        @(posedge clk);
        w_data = data;
        wr_uart = 1;
        @(posedge clk);
        wr_uart = 0;
    end
    endtask

  task receive_byte;
        begin
          // wait(rx_empty == 0);
            @(posedge clk);
            rd_uart = 1;
            @(posedge clk);
            rd_uart = 0;
        end
        endtask


endmodule
```

4.2 Waveform Analysis



```
// Test procedure
initial begin
    // Init
    clk = 0;
    rst = 1;
    wr_uart = 0;
    w_data = 8'h00;
    FINAL_VALUE = 10'd650;
    #50;

    // Release reset
    rst = 0;
    #20;
```

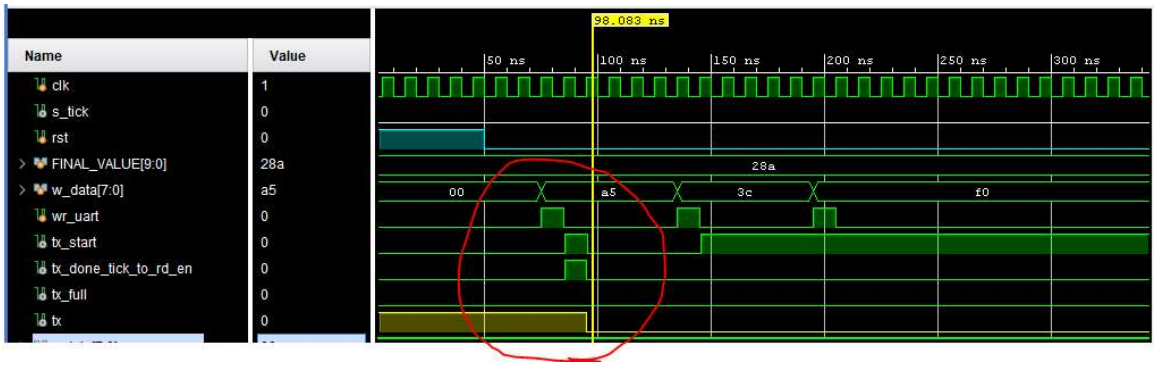
Baud Rate=9600

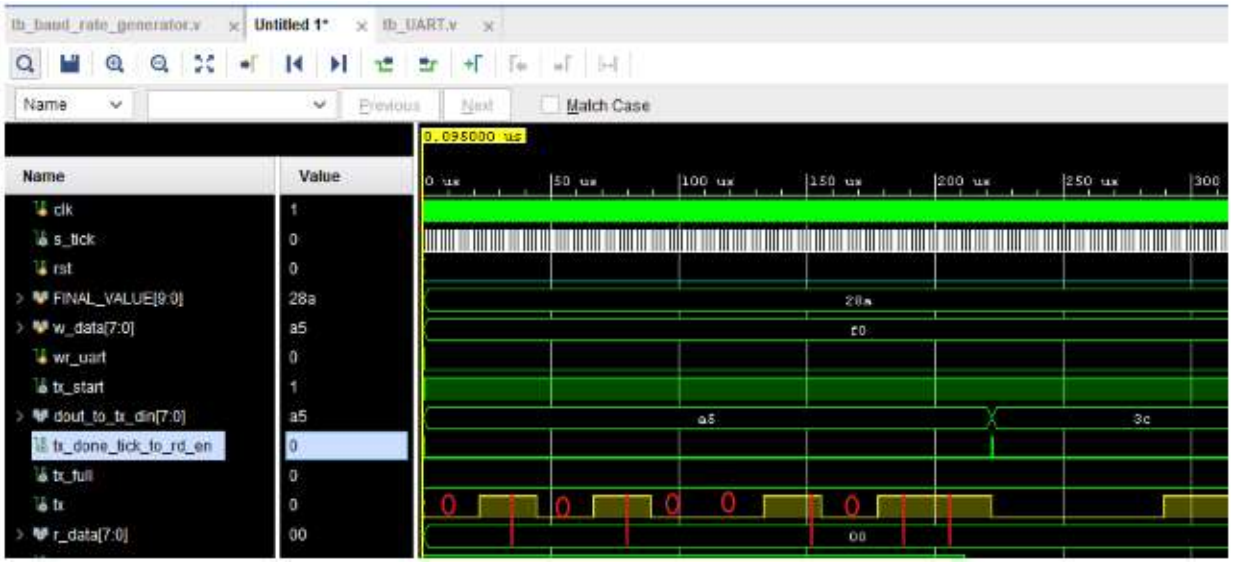
```
tb_UART.v
UART.srcs > sim_1 > new > tb_UART.v
23   module tb_UART;
83       initial begin
92           // Release reset
93           rst = 0;
94           #20;
95
96           // Send first byte
97           send_byte(8'hA5);
98           #50;
99
100          // Send second byte
101          send_byte(8'h3C);
102          #50;
103
104          // Send third byte
105          send_byte(8'hF0);
106
107          // Wait enough time for transmissio
108          #4000000;
```

```
tb_UART.v                                ESP-IDF: Search Error Hint
UART.srcs > sim_1 > new > tb_UART.v
116  #20;
119          $stop;
120      end
121
122      // Task to send byte
123      task send_byte(input [7:0] data);
124      begin
125          @(posedge clk);
126          w_data = data;
127          wr_uart = 1;
128          @(posedge clk);
129          wr_uart = 0;
130      end
131      endtask
132
133      task receive_byte;
134          begin
135              // wait(rx_empty == 0);
```

Send to tx_fifo 5A , 3C , F0

Once tx_fifo not Empty tx_start take action and dout_to_tx_din[7:0] and send data over tx

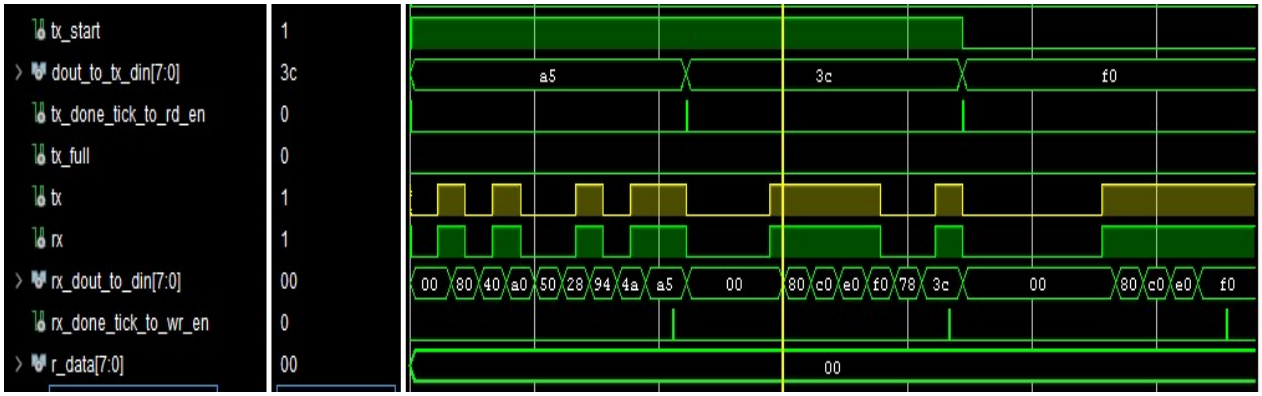




8’hA5 =8’b10100101

Tx=start_bit(0)+10100101 +stop_bit(1)

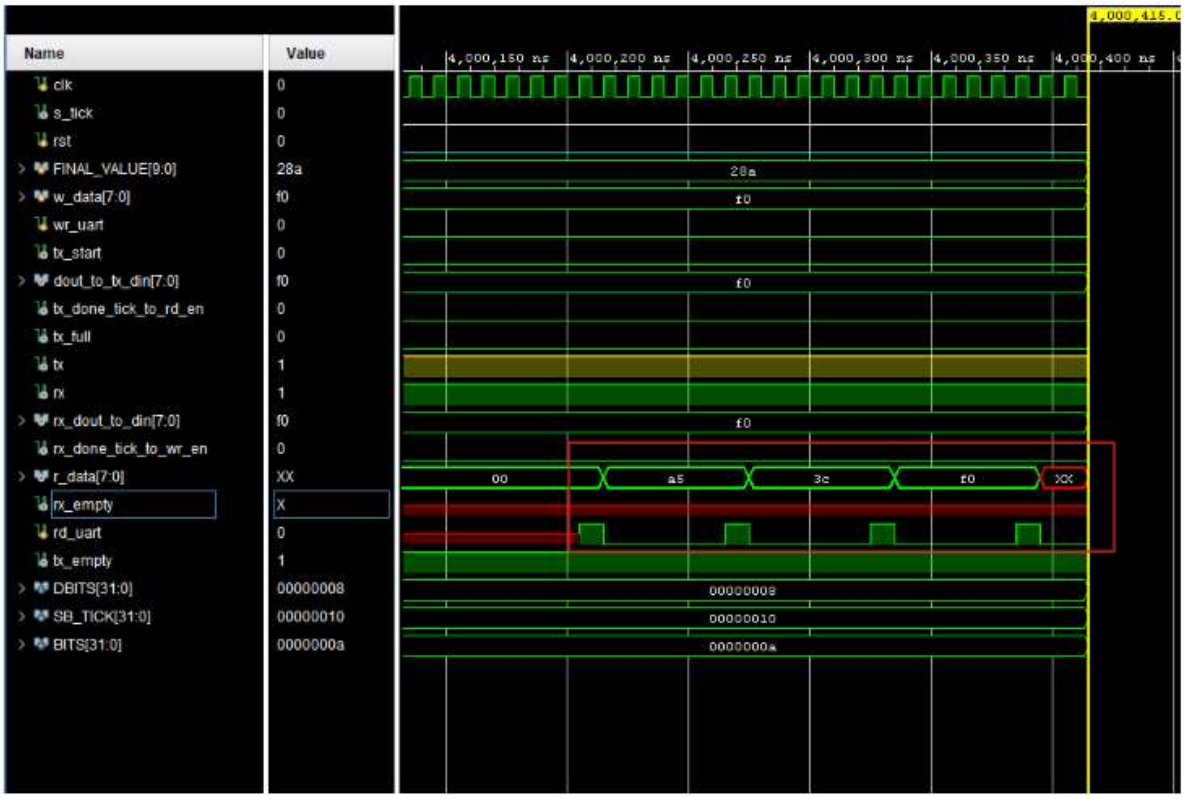
At same time rx =tx



Rx receive the data and store it at rx_fifo

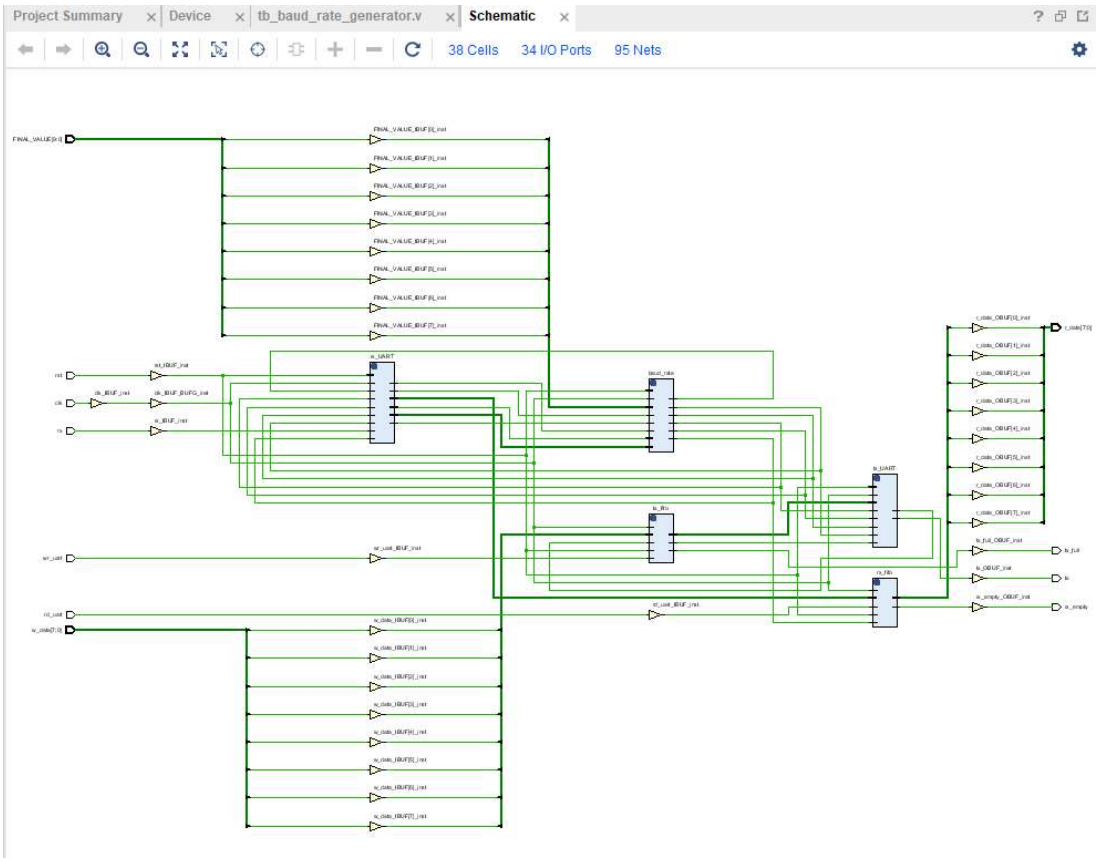
```
        // Send third byte
        send_byte(8'hF0);

        // Wait enough time for transmissio
        #4000000;
receive_byte;
#50;
receive_byte;
#50;
receive_byte;
#50;
receive_byte;
#20;


        $stop;
```

```
        wr_uart = 0;
    end
    endtask

    task receive_byte;
        begin
          // wait(rx_empty == 0);
            @(posedge clk);
            rd_uart = 1;
            @(posedge clk);
            rd_uart = 0;
        end
        endtask


endmodule
```

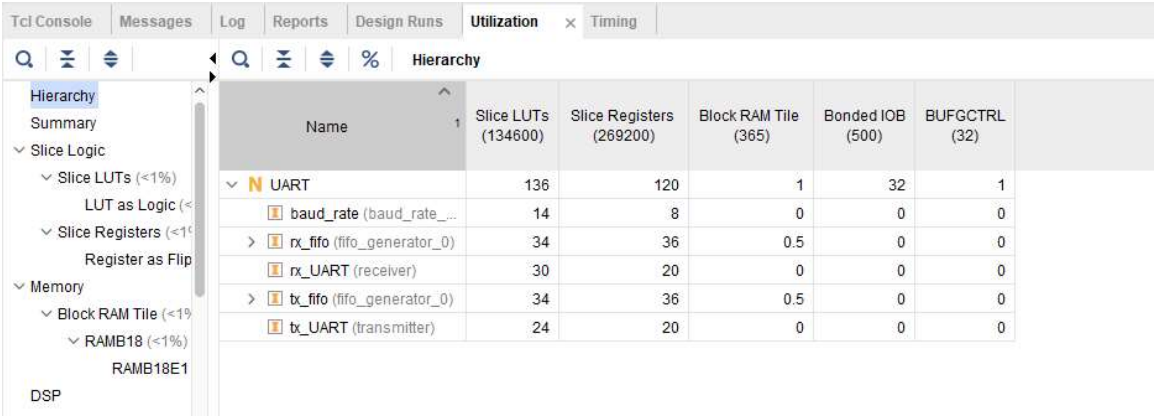


5. Synthesis

5.1 Synthesis



5.2 Resource Utilization



Name	Slice LUTs (134600)	Slice Registers (269200)	Block RAM Tile (365)	Bonded IOB (500)	BUFGCTRL (32)
UART	136	120	1	32	1
baud_rate (baud_rate_...	14	8	0	0	0
rx_fifo (fifo_generator_0)	34	36	0.5	0	0
rx_UART (receiver)	30	20	0	0	0
tx_fifo (fifo_generator_0)	34	36	0.5	0	0
tx_UART (transmitter)	24	20	0	0	0

5.3 Timing Report

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	5.979 ns	Worst Hold Slack (WHS):	0.130 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	271	Total Number of Endpoints:	271	Total Number of Endpoints:	125

All user specified timing constraints are met.

6. Results and Discussion

The UART system successfully transmitted and received data at different baud rates. FIFO ensured smooth data flow without data loss. The modular verification approach simplified debugging and improved reliability.

7. Conclusion

This project demonstrated a modular UART implementation in Verilog HDL. Future work may include parity bit support, error detection, and extension for multi-channel UARTs.