Problem

Unable to read out data from receiver.

Verilog Code

```
//UART MODULE

module uart_rx #(

        parameter DBIT = 8,     // # data bits

                SB_TICK = 16  // # stop bit ticks

                                                  )

          (reset, clk, rx, count, rx_dout, rx_done_tick);

   input reset; //reset

   input clk;  //clock

   input rx;   //receiver line

   input [4:0] count;  //count value for baud rate gen

   output [7:0] rx_dout;  //receiver output

   output rx_done_tick;   //receiving done status signal


   wire tick; //connect tick signal of baud rate generator and receiver


         //baud rate generator instance

         timer_circuit   baud_rate_gen(

         .clk(clk),    //clock

         .reset_n(reset),  //reset

         .enable(1'b1),  //enable

         .count_value(count),  //count value

         .tick_signal(tick)   //tick
         );

         //receiver instance
         uart_rx_module  #(.DBIT(DBIT), .SB_TICK(SB_TICK))receiver(
         .clk(clk),
         .reset_n(reset),
         .tick_signal(tick),
         .rx(rx),
         .rx_done_tick(rx_done_tick),
         .rx_dout(rx_dout)
         );


endmodule

//Baud Rate Generator
//A counter is used as timer circuit to generate tick signal
module timer_circuit(
   input clk,
   input reset_n,
   input enable,
   input [4:0] count_value,
   output tick_signal
   );

   reg [4:0] Q_reg, Q_next;
```

```
    always @(posedge clk, negedge reset_n)
    begin
        if (~reset_n)
            Q_reg <= 'b0;
        else if(enable)
            Q_reg <= Q_next;
        else
            Q_reg <= Q_reg;
    end

    //Generates a tick when Q_reg reaches count_value
    assign tick_signal = Q_reg == count_value;
    // Next state logic
    always @(*)
        Q_next = tick_signal? 1'b0: Q_reg + 1;



endmodule

//receiver module
module uart_rx_module
    #(parameter DBIT = 8,    // # data bits
                SB_TICK = 16 // # stop bit ticks
    )
    (
  input clk, //clock
                  input reset_n,   //reset
  input rx, //receiver line
  input tick_signal, //tick_signal
  output reg rx_done_tick,  //status signal
  output [DBIT-1:0] rx_dout //data output
);

     //local parameters for states
localparam  idle = 0, start = 1,
            data = 2, stop = 3;

reg [1:0] state_reg, state_next; //keep the track of states
reg [3:0] s_reg, s_next;              // keep track of the baud rate ticks (16 total)
reg [2:0] n_reg, n_next; // keep track of the number of data bits recieved
reg [DBIT - 1:0] b_reg, b_next;         // stores the recieved data bits

// State and other registers
always @(posedge clk, negedge reset_n)
begin
  if (reset_n)
  begin
    state_reg <= idle;
    s_reg <= 0;
    n_reg <= 0;
    b_reg <= 0;
    end
    else
    begin
        state_reg <= state_next;
        s_reg <= s_next;
        n_reg <= n_next;
        b_reg <= b_next;
    end
end

// Next state logic
always @(*)
begin
    state_next = state_reg;
    s_next = s_reg;
    n_next = n_reg;
    b_next = b_reg;
    rx_done_tick = 1'b0;
    case (state_reg)
    //idle state
        idle:
            if (~rx) // 0 represents start bit, when rx receives 0
            begin
                s_next = 0;  //s_next starts coounting the tcks
                state_next = start;    //receiver goes in start state
            end
```

```
//start state
start:
   if (tick_signal)    // when s_reg receives 7 tick after
      if (s_reg == 7) //7 ticks denotes start of data transmission
      begin
         s_next = 0;  //s_next is set to zero
         n_next = 0; //n_next is initialised
         state_next = data; // receiver goes into data transmission state
      end
      else
      s_next = s_reg + 1; //receiver keeps counting till it receive 7 ticks
//data transfer state
data:
   if (tick_signal) //data is taken as input for every 15 ticks
      if(s_reg ==15) //when s_reg counts 15 tick data is written in  the 'b
register'
      begin
         s_next = 0; //s_next is again set to 0 for next bit
         b_next = {rx, b_reg[DBIT - 1:1]};  //right sihft opertaion to write
data
         if (n_reg == (DBIT - 1)) // when n_reg =7 this means the register is
full
            state_next = stop;   //receiver moves to stop state
         else                  //else the count of 'n registers'
            n_next = n_reg + 1;  //increase after every write
      end
      else
```

```
                s_next = s_reg + 1;      //counts number of ticks
           //stop state
           stop:
              if (tick_signal)  //receiver stops after
                 if(s_reg == (SB_TICK - 1)) //receiving 15 ticks (stop bit ticks)
                 begin
                    rx_done_tick = 1'b1; //status signal of data received
                    state_next = idle;   // receiver once again goes to idle
                 end
                 else
                    s_next = s_reg + 1;
           //default state
           default:
              state_next = idle;
        endcase
     end

     // output logic
     assign rx_dout = b_reg;
endmodule
```

Testbench

```
module uart_rx_fifo_tb_v;

        // Inputs
```

```
reg reset;

reg clk;

reg rx;

reg [4:0] count;

// Outputs

wire [7:0] rx_dout;

wire rx_done_tick;

integer i=0;  //integer for loop

reg [7:0] TxData; //register for data transmission

// Instantiate the Unit Under Test (UUT)

uart_rx uut (

        .reset(reset),

        .clk(clk),

        .rx(rx),

        .count(count),

        .rx_dout(rx_dout),

        .rx_done_tick(rx_done_tick)

);

always begin

        clk = 1'b0;

        #5;

        clk =1'b1;

        #5;

        end


// Initial block for testbench stimulus
```

```
initial begin
    // Initialize inputs
                  rx =1; //rx line is by default at 1
                 count = 'd5; //counter value set at 5
                  reset = 1'b0;
                  #10;
                  reset= 1'b1;
                  #10;
                  reset= 1'b0;
                  #10;
                  TxData =8'b01010101;  //data to be transfered
         end
         initial begin
    //stimulus
         #5;
    @(negedge clk) rx=0; //start bit
                  #100;
                  for (i = 0; i < 8; i=i+1)
                  begin
                                #100;
        @(negedge clk)
        rx = TxData[i]; // Simulate received data
        @(negedge clk);
                  end
                  #100
                  @(negedge clk) rx=1; //stop bit
```

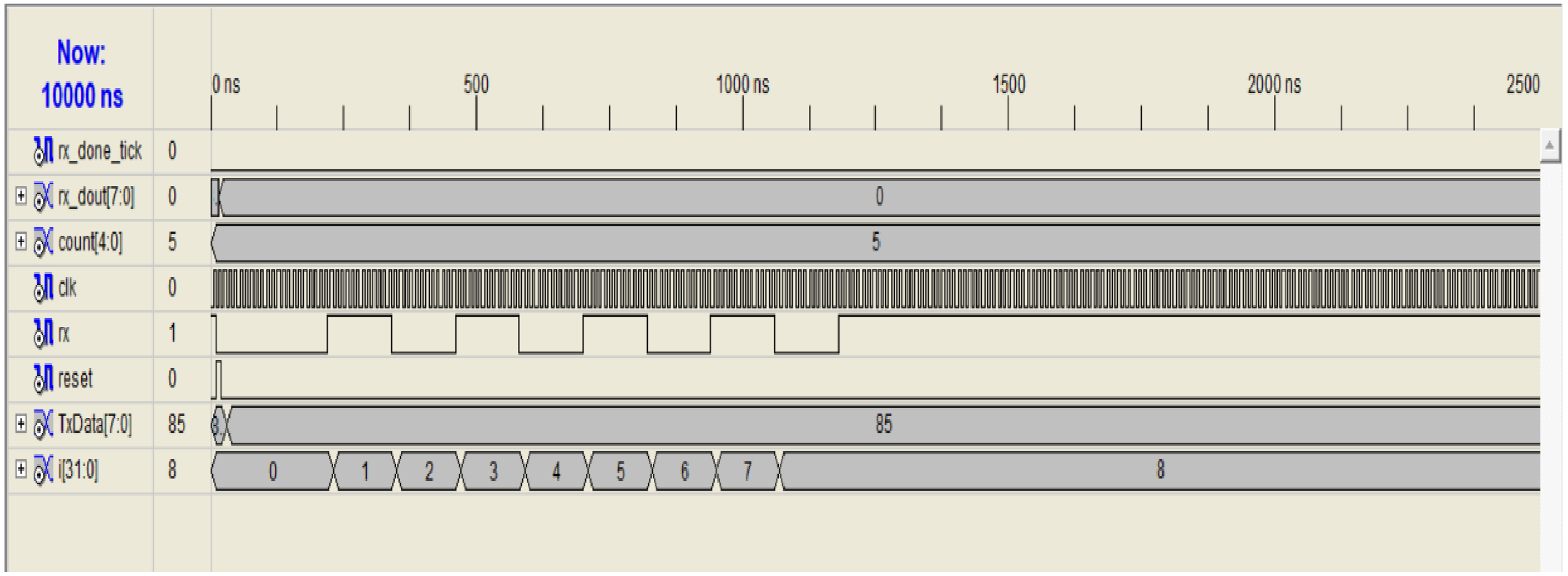
```
                #100;
        end
endmodule
```

Waveform

The waveform clearly shows that data was transmitted from *TxData* register to *RX* line, but it cannot be read out from *rx_dout* port.

Problem

Unable to read out data from receiver.

Now:
10000 ns
0 ns
500
1000 ns
1500
2000 ns
2500
rx_done_tick
rx_dout[7:0]
count[4:0]
clk
rx
reset
TxData[7:0]
i[31:0]