به نام خدا

درس: طراحی سیستم های دیجیتال

نیم‌سال تحصیلی دوم ۹۹۰۰

مدرس: بهارونـد

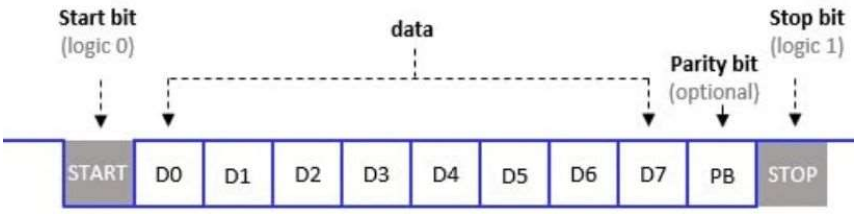
تمرین شماره: ۵	موعد تحویل: ۱۴۰۰/۳/۱۰ ساعت ۲۳:۵۵
نام و نام خانوادگی: سینا مظاهری	شماره دانشجویی: ۹۸۱۷۱۱۵۹

راه حل در همین فایل ارائه شود.

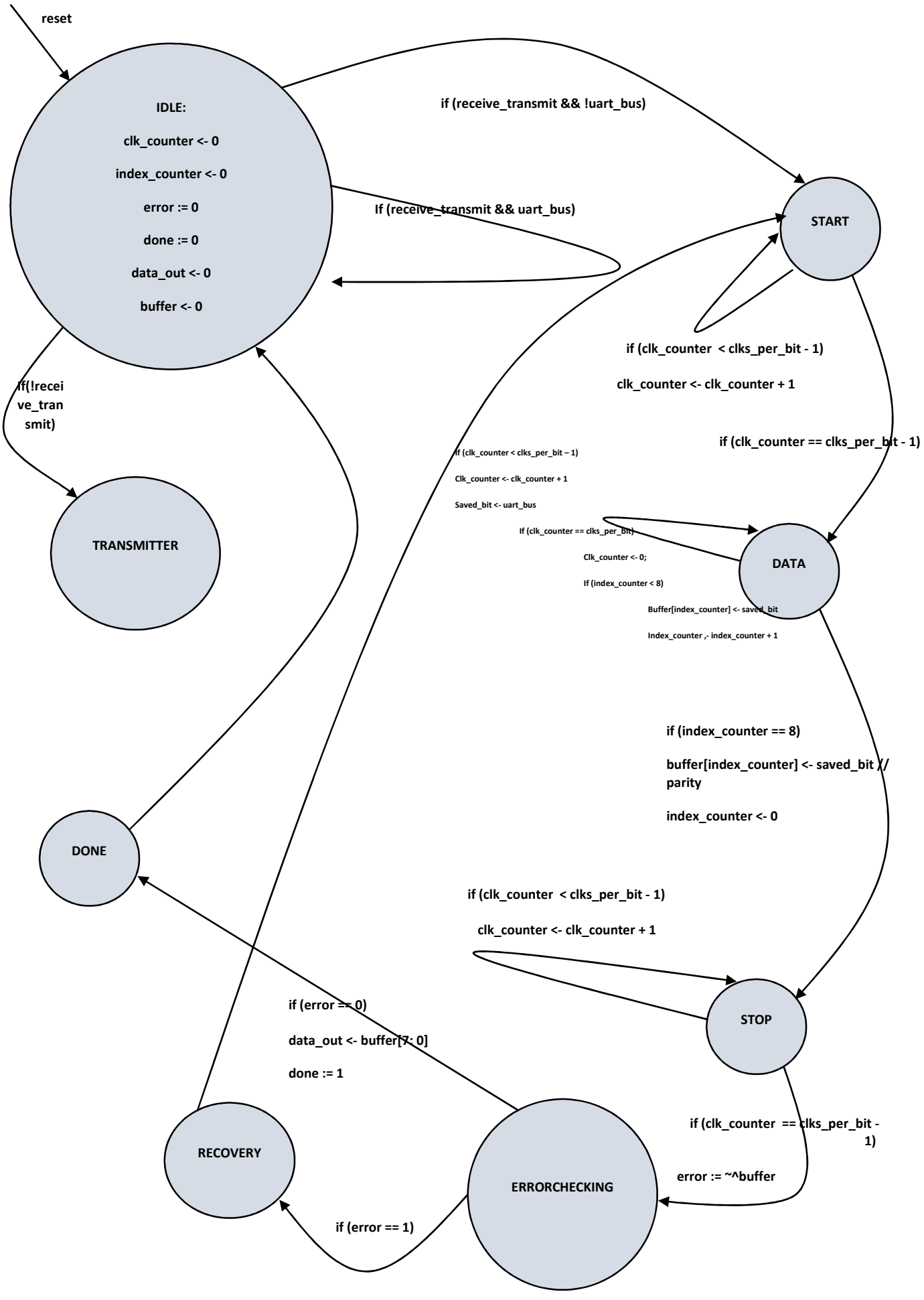
فایل به PDF تبدیل و در سایت بارگذاری شود.

عنوان تمرین

برای یک مدار **گیرنده** UART ابتدا ماشین حالت رسم نموده و سپس کد **رفتاری** متناظر آن را به صورت کامل طراحی کنید. در نظر داشته باشید داده‌های ورودی به UART دارای فرمت مطابق شکل زیر هستند. کلاک ورودی به مدار را uart_clock در نظر بگیرید و مدار را **پارامتری** طراحی کنید به نحوی که امکان کار با baudrate های مختلف در این گیرنده وجود داشته باشد. همچنین می‌خواهیم از پین ورودی این مدار گیرنده (با نام serial_data)، برای بخش فرستنده هم استفاده کنیم (یعنی خط ما bidirectional است و در هر لحظه فقط گیرنده یا فرستنده هستیم) و گذرگاه متصل به UART فقط یک سیم باشد؛ بدون نوشتن کد فرستنده، مکانیزمی در کد خود در نظر بگیرید که امکان این اتصال را فراهم کند. مدار بایستی دارای ریست آسنکرون باشد. طراحی به گونه ای باشد که، بیت توازن (پاریتی) را هنگام دریافت بایتها چک کنید و در صورت اشتباه بودن، مدار کار خود را از نو شروع کند.



به منظور طراحی یک واحد گیرنده **UART**، ابتدا ماشین حالت متناهی آن را رسم می نماییم. شکل **FSM** آن در زیر آمده است. توضیحات مورد نیاز نیز به دنبال آن نوشته شده است.



ماشین حالت متناهی فوق که در سوال قبل رسم شده، بدین گونه عمل می کند که ابتدا با ریست شدن آن ، به حالت اولیه ی **IDLE** می رود. در این حالت سیگنال های **error** و **done** صفر می شوند و در پالس بعدی، تمامی ثبات های مسیر داده به مقدار صفر می رسند.

```
module UART #(parameter clks_per_bit = 80)(
    input wire [7: 0] serial_data,
    input wire uart_clk,
    input wire reset,
    input wire receive_transmit,
    output reg done,
    inout wire uart_bus,
    output reg error,
    output reg [7: 0] data_out
    );

    reg [$clog2(clks_per_bit) - 1: 0] r_clock_counter;
    reg [8: 0] buffer;
    reg [4: 0] r_index_counter;

    reg [2: 0] current_state;
    reg saved_bit;
    localparam IDLE = 3'b000, START = 3'b001, DATA = 3'b010, STOP = 3'b011, ERRORCHECKING = 3'b100, DONE = 3'b101, TRANSMITTER = 3'b110, RECOVERY = 3'b111;

    always @(posedge uart_clk or negedge reset)
        begin
            if (!reset)
                begin
                    saved_bit <= 0;
                    buffer <= 0;
                    current_state <= IDLE;
                    r_clock_counter <= 0;
                    r_index_counter <= 0;
                    error <= 0;
                    data_out <= 0;
                    done <= 1'b0;
                end
```

این مدار دارای قابلیت سوییچ بین حالت فرستنده و گیرنده می باشد بدین صورت که با فعال کردن سیگنال **receive_transmit** می توان نوع گیرنده و در غیر اینصورت نوع فرستنده ا برای آن انتخاب کرد اما در این مدار صرفا یک حالت نمادین برای فرستنده پیاده سازی شده و فاقد هر گونه مدار کامل برای فرستنده می باشد زیرا هدف از طرح سوال تنها رسم مدار برای گیرنده بوده و مکانیزمی که بتواند این دو را از یکدیگر تشخیص دهد. همچنین یک ورودی ۸ بیتی به نام **serial_data** وجود دارد که در صورت کامل بودن مدار فرستنده می توان از طریق آن به ارسال داده اقدام کرد. همانطور که گفته شد در واقع حالت **TRANSMITTER** ، خود یک گراف ماشین حالت متناهی مجزا دارد که در صورتی که آن به طور کامل پیاده سازی شود، مدار قادر به ارسال اطلاعات نیز می شود. این مدار دارای ریست آسنکرون می باشد که با **negedge** شدن آن مدار به حالت اولیه، یعنی **IDLE** می رود.سیگنال **uart_clk** ، سیگنال ساعت مدار بوده و به مولد پالس متصل می شود. دو ثبات **error** و **data_out** هر کدام به ترتیب، خطا و داده دریافت شده را نشان می دهند. در واقع زمانی داده دریافت شده معتبر است که سیگنال **done** مقدار ۱ را به خود گرفته باشد. در صورتی که بیت توازن ، توازن فرد داده های ارسالی را بر هم زند، سیگنال **error** فعال می شود و مدار به حالت **RECOVERY** می رود و سپس در پالس بعدی به حالت **START** که نقطه ی شروع دریافت داده است مراجعه می کند. حال که کلیاتی را در مورد مدار بیان کردیم به سراغ تشریح حالات مختلف می رویم. نکته ی دیگری که قابل توجه است آن است که مدار به شکل پارامتری طراحی شده که با **clk_per_bit** های مختلف، بسته به اینکه فرکانس کاری مدار و ارسال اطلاعات چه باشد، خود را تنظیم کند.

```
else
    begin
        case (current_state)
            IDLE:
                begin
                    r_clock_counter <= 0;
                    r_index_counter <= 0;
                    error <= 0;
                    data_out <= 0;
                    done <= 1'b0;
                    if (receive_transmit)
                        begin
                        buffer <= 0;
                            if (!uart_bus)
                                    current_state <= START;
                            else
                                    current_state <= IDLE;
                        end
                    else
                        begin
                                    current_state <= TRANSMITTER;
                                    buffer <= serial_data;
                        end
                end
```

پس از ریست شدن مدار، مادامی که در حالت **IDLE** قرار دارد، در صورتی که سیگنال ارتباطی (سیم گیرنده) و حالت گیرنده بودن که توسط **receive_transmit** در مدار مشخص می گردد، فعال باشد، مدار به حالت شروع می رود و در غیر اینصورت در همان حالت می ماند.

```
            end
        START:
            begin
                 if (r_clock_counter < clks_per_bit - 1)
                    begin
                        current_state <= START;
                        r_clock_counter <= r_clock_counter + 1;
                    end
                 else
                    begin
                        current_state <= DATA;
                        r_clock_counter <= 0;
                    end
             end
```

مادامی که در حالت شروع باشیم، شمارنده کلاک تعداد کلاک های بیت شروع یعنی صفر را می شمارد. تا نرخ ارسال اطلاعات را رعایت کند. پس از دریافت بیت شروع به حالت داده می رود.

```
                    DATA:
                      begin
                          if (r_clock_counter < clks_per_bit - 1)
                              begin
                                  saved_bit <= uart_bus;
                                  current_state <= DATA;
                                  r_clock_counter <= r_clock_counter + 1;
                              end
                          else
                              begin
                                  r_clock_counter <= 0;
                                  if (r_index_counter < 8)
                                      begin
                                          current_state <= DATA;
                                          buffer[r_index_counter] <= saved_bit;
                                          r_index_counter <= r_index_counter + 1;
                                      end
                                  else
                                      begin
                                          buffer[r_index_counter] <= saved_bit;
                                          current_state <= STOP;
                                          r_index_counter <= 0;
                                      end
                              end
                        end
```

مادامی که در حالت دریافت داده باشیم، ابتدا هر بیت داده را از کم ارزش به پرارزش با رعایت توالی کلاک های نرخ ارسال اطلاعات دریافت می کنیم و پس از دریافت هر بیت، آن را در بافر ذخیره و در نهایت شمارنده کلاک را برای بیت بعدی صفر می کنیم. در نهایت بیت توازن را ذخیره نموده و به حالت توقف می رویم.

```
                    STOP:
                       begin
                            if (r_clock_counter < clks_per_bit - 1)
                                begin
                                    current_state <= STOP;
                                    r_clock_counter <= r_clock_counter + 1;
                                end
                            else
                                begin
                                    current_state <= ERRORCHECKING;
                                    error <= ~^buffer;
                                    r_clock_counter <= 0;
                                end
                       end
                    ERRORCHECKING:
                       begin
                            if (error)
                                current_state <= RECOVERY;
                            else
                                begin
                                done <= 1'b1;
                                data_out <= buffer[7: 0];
                                current_state <= DONE;
                                end
                       end
                    DONE: current_state <= IDLE;
                    RECOVERY:  current_state <= START;
            endcase
        end
    end

endmodule
```

در حالت توقف نیز، بیت پایان را مانند حالت شروع دریافت نموده و پس از **xnor** کردن تمامی داده ها به منظور بررسی توازن فرد بودن به حالت بررسی خطا می رویم. در صورتی که توازن زوج باشد، مدار داده ارسالی را خطا تشخیص می دهد و به حالت بازیابی و سپس به حالت شروع می رود. در غیر اینصورت نیز داده مورد نظر در ثبات خروجی ذخیره شده و سیگنال **done** یک می شود که نشان دهنده آماده بودن داده ی دریافتی است.

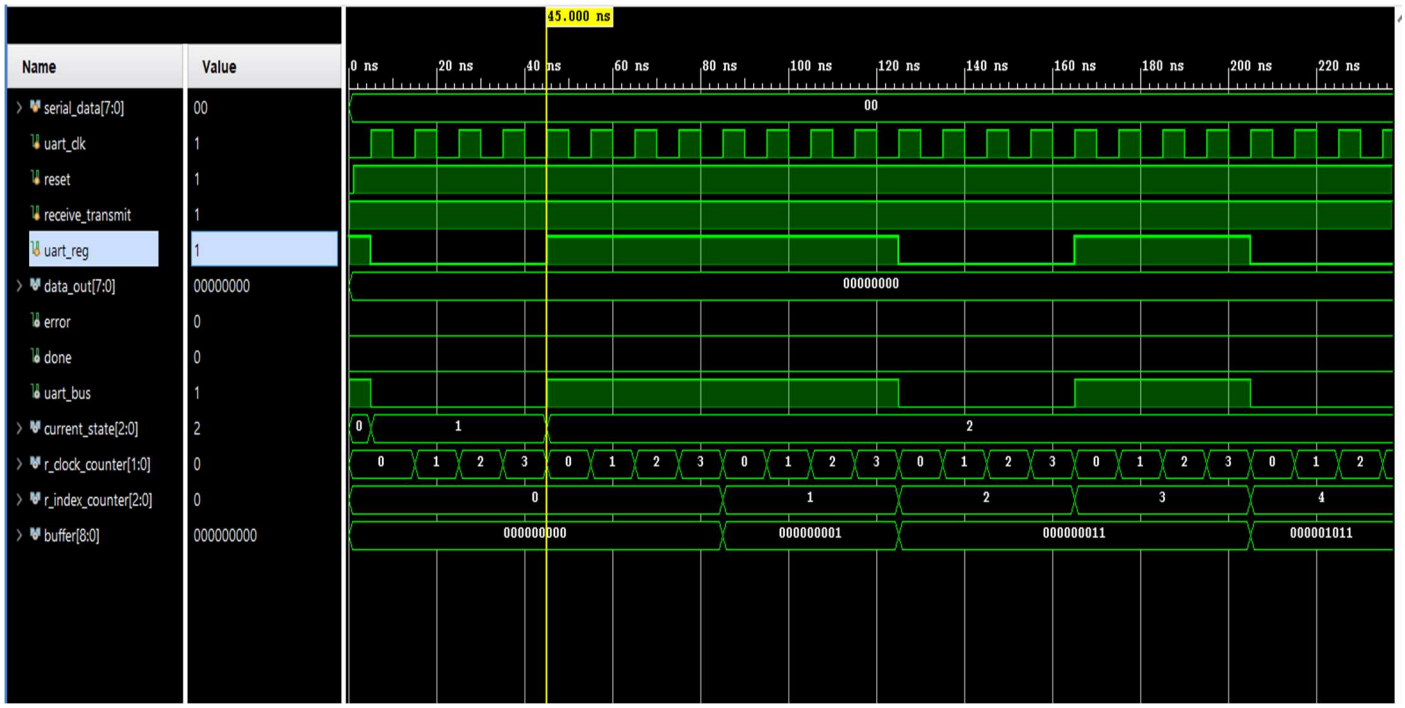
شبیه سازی

به منظور شبیه سازی سیستم پیاده سازی شده ابتدا یک طرح تست مانند شکل زیر برای آن می نویسیم و داده ی زیر که به صورت ۰۱۱۰۱۰۱۱ می باشد را می فرستیم. توجه داریم که با ارزشترین بیت همان بیت توازن می باشد که در اینجا مقدارش صفر است.

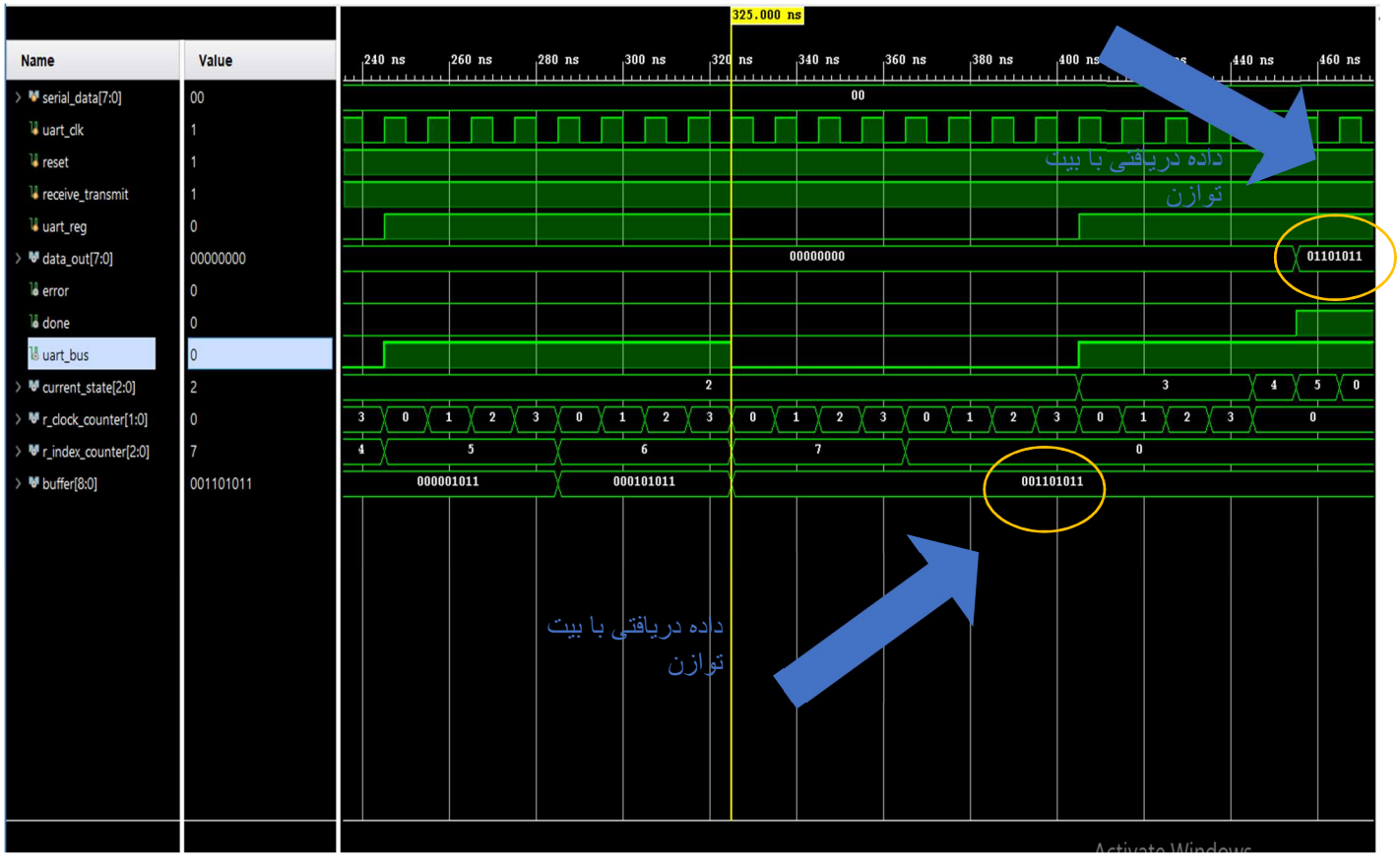
```
        end

    //Data = 01101011, Parity = 0, Transmited Data: 001101011 (Odd parity)
    wire [2: 0] current_state;
    wire [1: 0] r_clock_counter;
    wire [2: 0] r_index_counter;
    wire [8: 0] buffer;
    assign buffer = uart0.buffer;
    assign current_state = uart0.current_state;
    assign r_clock_counter = uart0.r_clock_counter;
    assign r_index_counter = uart0.r_index_counter;
    initial
       begin
           uart_reg = 1'b1;
           serial_data = 8'b0;
           receive_transmit = 1'b1;
           #1 reset = 1'b1;
           #4 uart_reg = 1'b0; // Start Bit
           #40 uart_reg = 1'b1; // Data Bits
           #40 uart_reg = 1'b1;
           #40 uart_reg = 1'b0;
           #40 uart_reg = 1'b1;
           #40 uart_reg = 1'b0;
           #40 uart_reg = 1'b1;
           #40 uart_reg = 1'b1;
           #40 uart_reg = 1'b0;
           #40 uart_reg = 1'b0; // Parity Bit
           #40 uart_reg = 1'b1;// Stop Bit
       end

endmodule
```

شکل موج شبیه سازی شده به صورت زیر است.

شکل ۱


45.000 ns
Name
Value
serial_data[7:0]
00
uart_clk
1
reset
1
receive_transmit
1
uart_reg
1
data_out[7:0]
00000000
error
0
done
0
uart_bus
1
current_state[2:0]
2
r_clock_counter[1:0]
0
r_index_counter[2:0]
0
buffer[8:0]
000000000
000000001
000000011
000001011

شکل ۲


325.000 ns
Name
Value
serial_data[7:0]
00
uart_clk
1
reset
1
receive_transmit
1
uart_reg
0
data_out[7:0]
00000000
01101011
error
0
done
0
uart_bus
0
current_state[2:0]
2
r_clock_counter[1:0]
0
r_index_counter[2:0]
7
buffer[8:0]
001101011
000001011
000101011
001101011
داده دریافتی با بیت توازن
داده دریافتی با بیت توازن