DIGITAL SYSTEM DESIGN APPLICATION

EHB436E CRN: 11280

PROJECT2

Salih Ömer Ongün

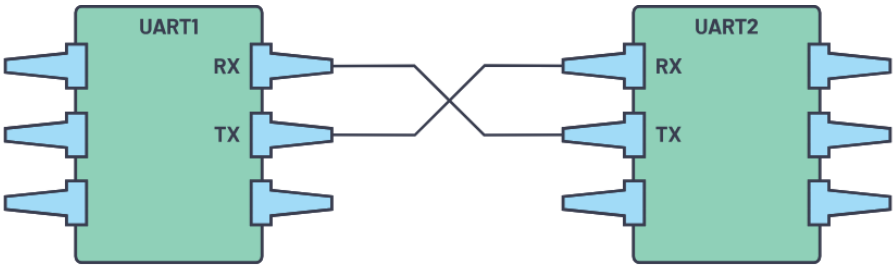
040220780

Salih Ömer Ongün
040220780

UART PROTOCOL

Communication protocols are very important in the use of electronic integrated circuits and devices. Certain rules must be followed for two or more electronic components to be understood and communicated correctly. For this reason, there are many protocols in communication systems. The UART protocol is one of them.

UART uses two wires to transmit and receive data. UART is a hardware communication protocol that uses asynchronous serial communication with configurable speed. Asynchronous means there is no clock signal to synchronize the output bits from the transmitting device going to the receiving end.[1]



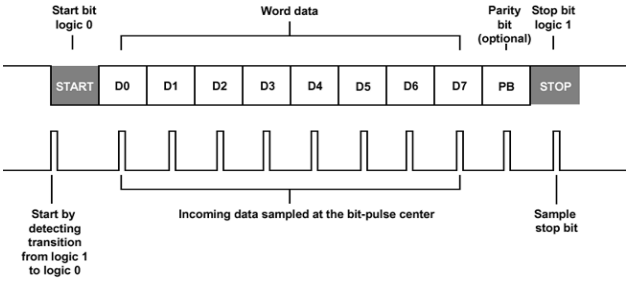
[1]

The clk signal is not used for synchronization of data in UART. Data is sent asynchronously. There is no data transmission in UART based on the clk signal, there is asynchronous transmission. Tx and rx process the data they obtain according to the specified baud rate. There are different baud rates. It is necessary to decide in advance which rate to use.

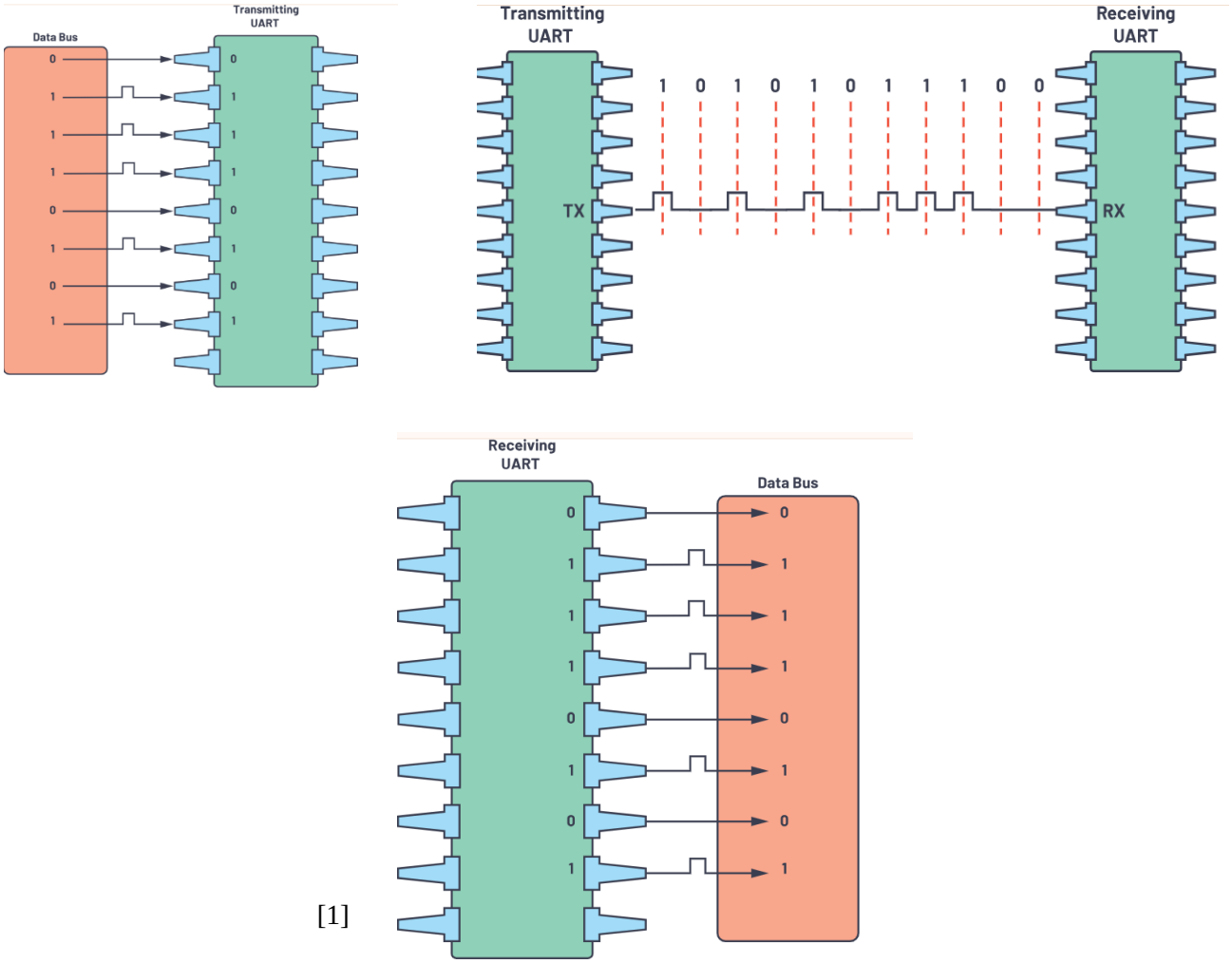
[1]

Normally, the logic 1 bit is always on. When the start bit becomes logic zero, the system starts working. It usually has 8 bits of data. In addition, the stop bit is sent at the end to stop the system. The stop bit is logic 1.

[2]



The transmitter receives data in parallel. It samples within itself with the baud rate frequency and gives parallel data to the output serially. The receiver receives the data serially. It samples the data with the baud rate frequency determined within itself and gives the serial data to the output in parallel.



[1]

Salih Ömer Ongün
040220780

ALGORITHM

I create four modules. These are uart_rx, uart_tx, baud_gen, uart_top. The uart_rx module acts as the receiver of the uart. The uart_tx module acts as the transmitter of the uart. It has a baud gen frequency divider structure. Therefore, it produces the baud rate we need and 8x equivalent of the baud rate. Uart_top is the top module where I combine all the modules.

Baud Rate Generator Module

This module produces the desired baud rate and 8x the equivalent of the baud rate, thanks to its parameterized structure from the internal 100 MHz clk signal. For this project, the baud rate should be 9600 and 115200.

Design Source

```
module baud_gen #(parameter [16:0]  freq= 115200)
(
    input clk,
    input rst,
    input baud_en,
    output count_8x_ready,
    output count_baud_ready
);
    localparam integer clk_freq = 100000000; // Internal clock frequency (100 MHz)
    reg count_8x_ready_reg,count_baud_ready_reg;
    reg [13:0] counter_8x;
    reg [2:0] counter_baud;
    always @(posedge clk) begin
        if(rst==1'b1) begin
            counter_baud <= 3'b0; // baud counter
            count_baud_ready_reg <= 0; // baud counter result
            counter_8x <= 13'b0; // 8x counter
            count_8x_ready_reg <= 0; // 8x counter result
        end
        else if (baud_en == 1) begin
            if(counter_8x ==((clk_freq/(freq*8))-1)) begin
                counter_8x <= 13'b0;
                count_8x_ready_reg <= 1;
                if(counter_baud == 7) begin
                    counter_baud <= 3'b0;
                    count_baud_ready_reg <= 1;
                end
                else begin
                    counter_baud <= counter_baud + 1;
                    count_baud_ready_reg <= 0;
                end
            end
            else begin
                counter_8x <= counter_8x + 1;
                count_8x_ready_reg <= 0;
                count_baud_ready_reg <= 0;
            end
        end
        else begin
            count_8x_ready_reg <= 0;
            counter_8x <= 13'b0;
            counter_baud <= 13'b0;
            count_baud_ready_reg <= 0;
        end
     end
    assign count_8x_ready = count_8x_ready_reg;
    assign count_baud_ready = count_baud_ready_reg;
endmodule
```

First, I defined our internal 100 MHz frequency for frequency division in the module. Then I created counters for both 8x and baud rate. Since 8x will have higher frequency and lower period, I calculated it first and gave it to the output. Then, when 8x becomes logical 1 every 8 times, I obtained the desired baud rate. For the operations to take place, that is, for the frequency division operation to take place, baud_en must be 1. If it becomes zero and then becomes one again, the counting process starts over. I used this in the receiver.

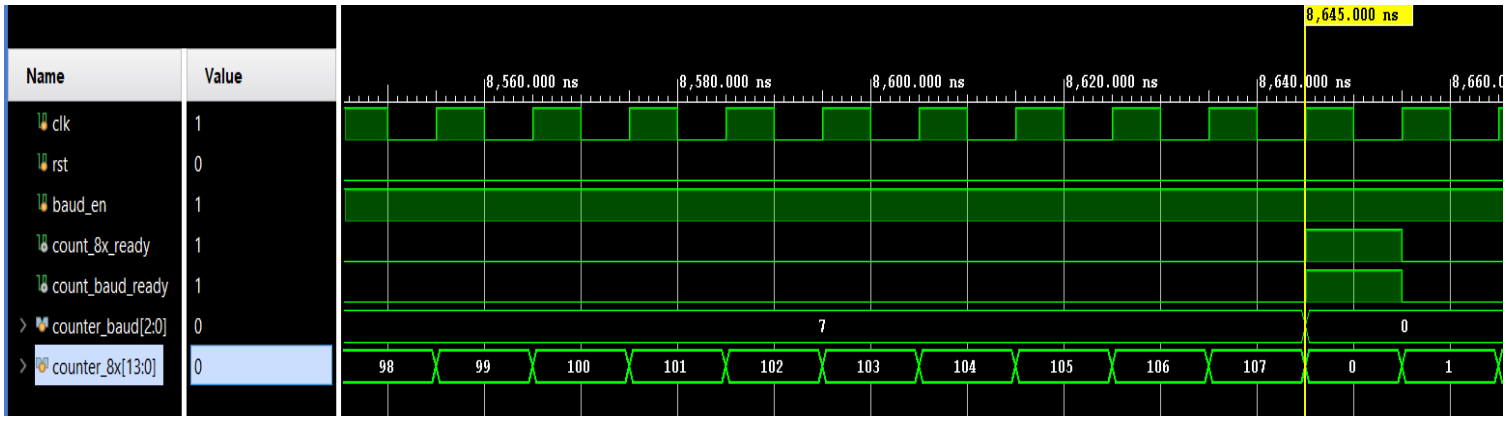
Simulation Source

```
module baud_gen_tb();

    // Parameters
    parameter [16:0] freq = 115200;

    // Testbench Signals
    reg clk = 1'b0;     // Clock
    reg rst = 1'b0;     // Reset
    reg baud_en = 1'b0; // Enable signal
    wire count_8x_ready; // 8x clock ready
    wire count_baud_ready; // Baud clock ready

    // Instantiate the DUT (Device Under Test)
    baud_gen #(.freq(freq)) uut (
        .clk(clk),
        .rst(rst),
        .baud_en(baud_en),
        .count_8x_ready(count_8x_ready),
        .count_baud_ready(count_baud_ready)
    );

    // Generate a 100 MHz clock (period = 10 ns)
    always begin
        #5 clk = ~clk;
    end

    initial begin
        rst = 1;
        #10;
        rst = 0;

        baud_en = 1;

        #11000;

        baud_en = 0;

        #200;
        $finish;
    end

endmodule
```

Simulation Waveform

$$baud\_div = \frac{10^8}{freq} - 1$$

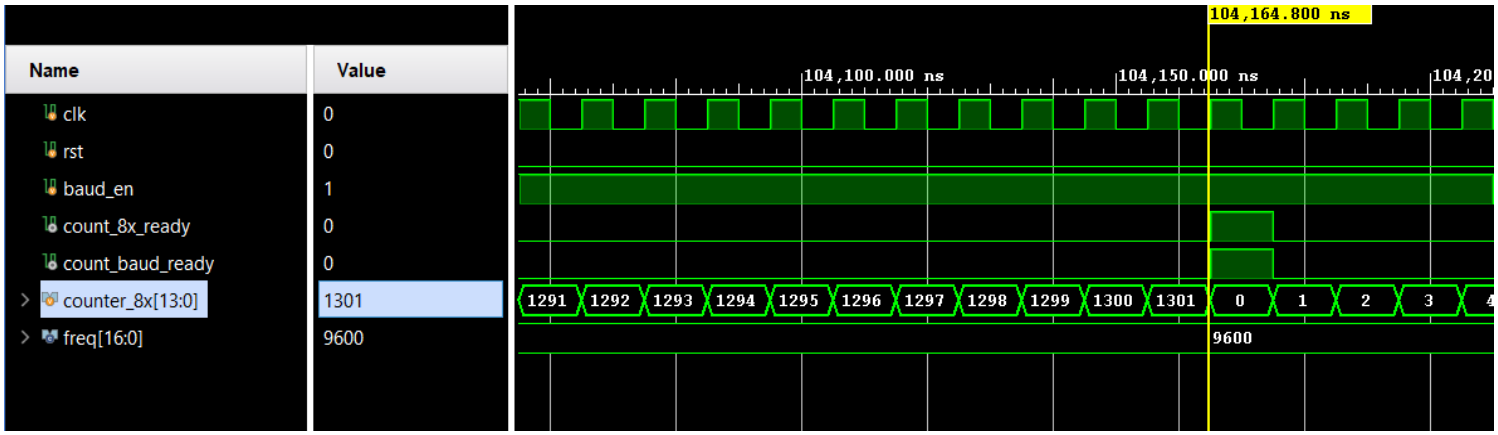
First, I will simulate the baud rate for 115200 Hz frequency.



As you can see in the picture, the code is working correctly.

$$baud\_div = \frac{10^8}{115200 * 8} - 1 = 107$$

Second,I will simulate the baud rate for 9600 Hz frequency.



$$baud\_div = \frac{10^8}{9600 * 8} - 1 = 1301$$

As you can see in the picture, the code is working correctly.

Salih Ömer Ongün
040220780

UART TRANSMITTER

DESIGN SOURCE

```
module uart_tx #(parameter [16:0]  freq= 115200)
(
    input clk,
    input rst,
    input [7:0] d_in,
    input tx_en,
    output seri_out,
    output reg start,
    output reg busy,
    output done
);


    wire count_baud_ready,out_of_use;
    reg done_reg, seri_out_reg,baud_en;
    reg [2:0] bit_in;
    reg [7:0] buffer;
    localparam IDLE = 2'b00;
    localparam START = 2'b01;
    localparam DATA = 2'b10;
    localparam STOP = 2'b11;
    reg [1:0] state;

    baud_gen #(.freq(freq)) divider
    (
        .clk(clk),
        .rst(rst),
        .baud_en(baud_en),
        .count_8x_ready(out_of_use),
        .count_baud_ready(count_baud_ready)
    );

    always @(posedge clk) begin

        if (rst == 1'b1) begin
            state <= IDLE;
            done_reg <= 0;
            start <= 0;
            busy <= 0;
            seri_out_reg <= 1;
            baud_en <= 0;
        end
        else begin
            case(state)
                IDLE: begin
                    done_reg <= 0;
                    seri_out_reg <= 1;
                    baud_en <= 0;
                    buffer <= d_in;
                    if (tx_en == 1) begin
                        state <= START;
                        bit_in <= 0;
                    end
                end

                START: begin
                    start <= 1;
                    seri_out_reg <= 0;
                    baud_en <= 1;
                    if(count_baud_ready == 1) begin
                        state <= DATA;
                    end
                end
```

```
                DATA: begin
                    start <= 0;
                    busy <=1;
                    seri_out_reg <= buffer[0];
                    if(count_baud_ready == 1) begin
                        if(bit_in == 7) begin
                            state <= STOP;
                            bit_in <= 0;
                        end
                        else begin
                            buffer[6:0] <= buffer[7:1];
                            bit_in <= bit_in + 1;.
                        end
                    end
                end

                STOP: begin
                    seri_out_reg <= 1;
                    if(count_baud_ready ==1) begin
                        state <= IDLE;
                        baud_en <= 0;
                        busy <=0;
                        done_reg <= 1;
                    end
                end

                default: begin
                    state <= IDLE;
                    done_reg <= 0;
                    seri_out_reg <= 1;
                    baud_en <= 0;
                end

            endcase
        end
    end
    assign done = done_reg;
    assign seri_out = seri_out_reg;
endmodule
```

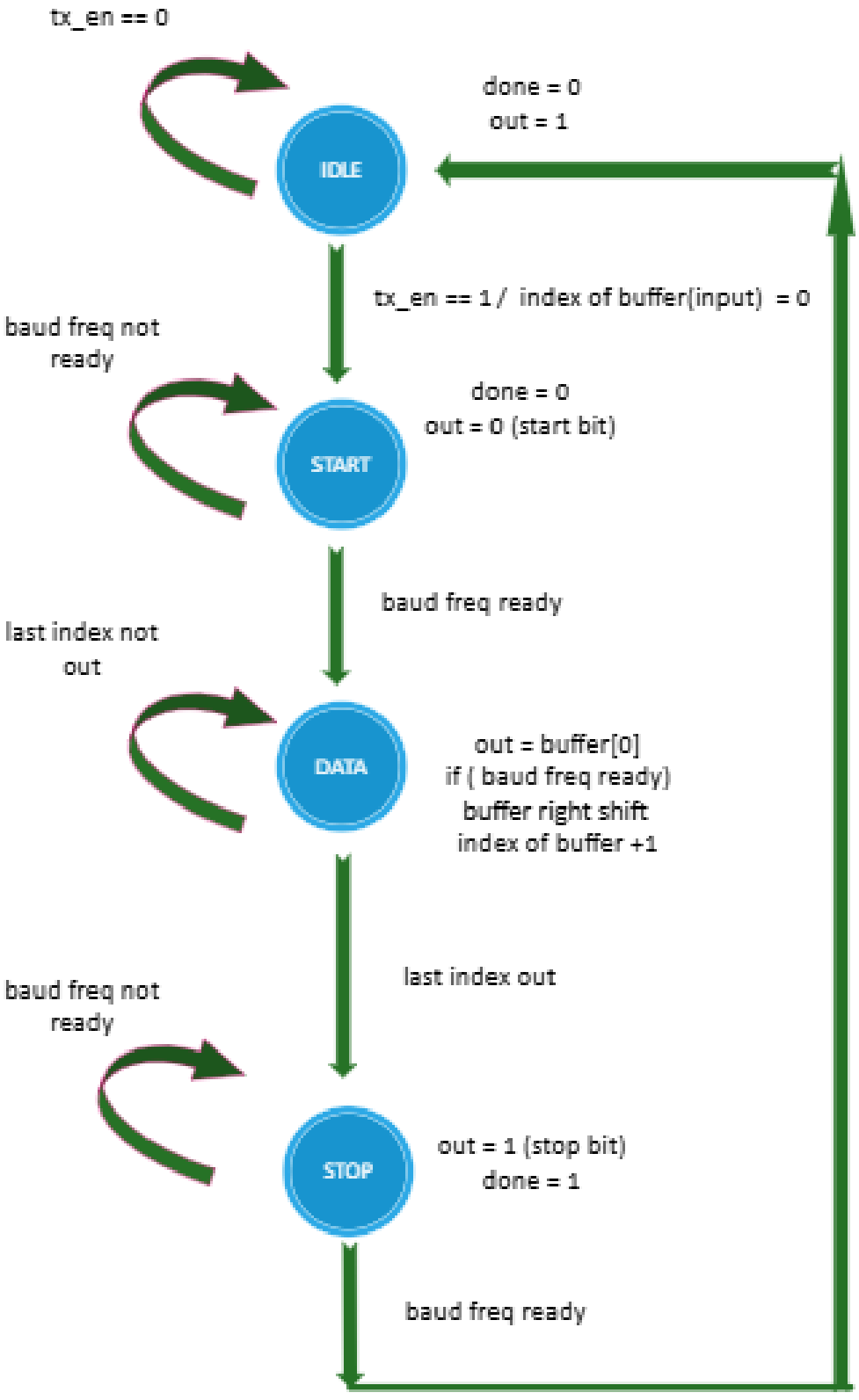
Since I made the transmitter before writing the top module, I had to instantiate the baud_gen module in the transmitter. When I made the top module, I connected them all in the top module.

Our transmitter normally remains in IDLE state. If we make each enable signal logical 1, the system starts working and goes to the START state.

In the START state, our output becomes logical 0 to create the start bit during 1 baud frequency. Then it goes to DATA state. In DATA, first the first value of the input is given to the output and waits for 1 baud frequency. Then, with the expression “buffer[6:0] <= buffer[7:1];”, I shift the input that I assigned to the buffer one bit to the right and give it to the output. It also waits for 1 baud frequency. This process goes until the last bit of the input. Then it goes to STOP state.

In the stop state, the stop bit, logical 1, is given to the output during 1 baud frequency. Finally, the done output becomes 1 and the process ends.

Finite State Machine Transmitter



Simulation Source

```
module uart_tx_tb;

    parameter [16:0] freq = 115200;
    reg clk = 1'b0;
    reg rst = 1'b0;
    reg [7:0] d_in;
    reg tx_en = 1'b0;
    wire seri_out;
    wire done;

    integer file, r;
    reg [7:0] stimulus_data;
    integer i;

    uart_tx #(.freq(freq)) uut (
        .clk(clk),
        .rst(rst),
        .d_in(d_in),
        .tx_en(tx_en),
        .seri_out(seri_out),
        .done(done)
    );

    // Saat sinyali üretimi
    always #5 clk = ~clk; // 10ns periodlu (100 MHz)

    initial begin

        file = $fopen("stimulus2.txt", "r");
        if (file == 0) begin
            $display("Error: Cannot open stimulus2.txt");
            $finish;
        end

        rst = 1;
        d_in = 8'b0;
        tx_en = 0;
        #8000;

        rst = 0;

        for (i = 0; i < 4; i = i + 1) begin
            r = $fscanf(file, "%b\n", stimulus_data);

            if (r == 1) begin
                d_in = stimulus_data;
                tx_en = 1;
                #10;
                tx_en = 0;

                wait (done == 1);
                #15000;
            end
        end
        $fclose(file);
        $finish;
    end

endmodule
```

I wrote python code to generate random numbers. In the simulation, I took 4 data from the file I created in python and checked it.

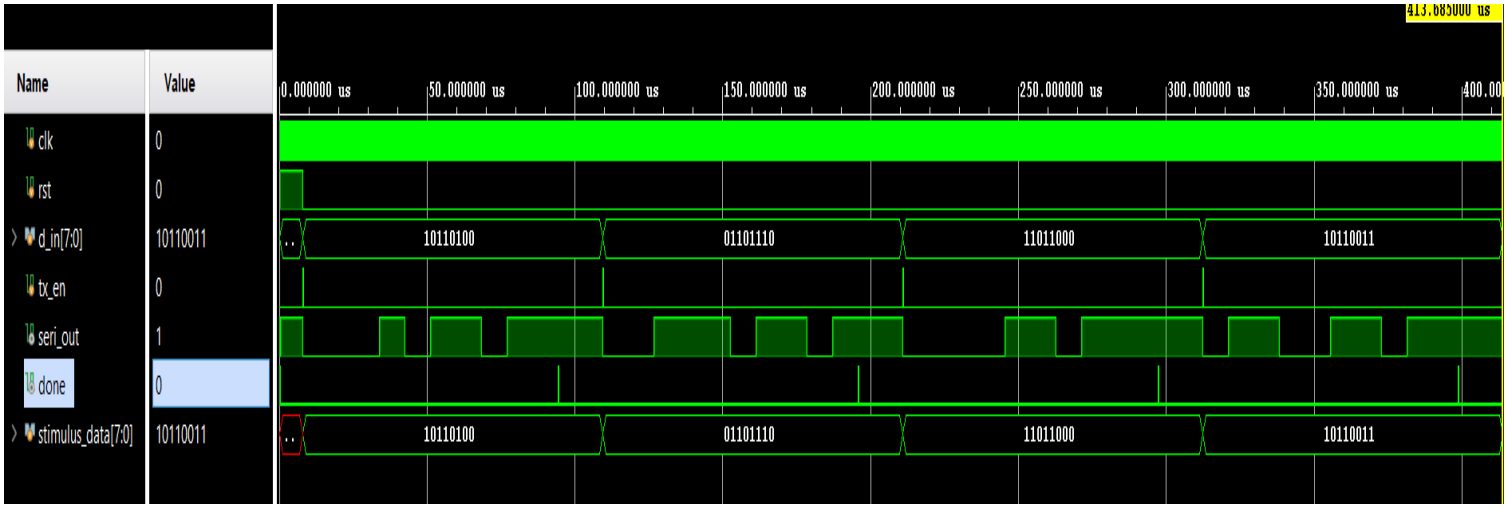
```python
import random

def gen_binary():
    with open("stimulus2.txt","w") as f:
        for k in range(0,4):
            number = random.getrandbits(8)
            bin_num = format(number, "0b")
            print(len(bin_num))
            if len(bin_num)<8:
                for i in range(0,8-len(bin_num)):

                    bin_num = "0" + str(bin_num)
            f.write(bin_num +"\n")
            print("----------------")
```

Phyton Code

```
10110100
01101110
11011000
10110011
```

File Content

Simulation Waveform



As seen in the picture, the code works correctly. I took 4 different 8-bit random numbers from the file. I tested it and serial_out gave the correct output.

Salih Ömer Ongün
040220780

UART RECEIVER

DESIGN SOURCE

```
module  uart_rx #(parameter [16:0]  freq= 115200)
(
    input clk,
    input rst,
    input d_in,
    input rx_en,
    output [7:0] d_out,
    output reg start,
    output reg busy,
    output done
);
    reg [3:0] count_ones;
    reg [3:0] count_zeros;
    reg [2:0] counter_halfperiod;
    wire count_baud_ready,count_8x_ready;
    reg done_reg,baud_en;
    reg [7:0] d_out_reg;
    reg [3:0] bit_in;
    localparam IDLE = 2'b00;
    localparam START = 2'b01;
    localparam DATA = 2'b10;
    localparam STOP = 2'b11;
    reg [1:0] state;

    baud_gen #(.freq(freq)) divider
    (
        .clk(clk),
        .rst(rst),
        .baud_en(baud_en),
        .count_8x_ready(count_8x_ready),
        .count_baud_ready(count_baud_ready)
    );

    always @(posedge clk) begin

        if (rst == 1'b1) begin
            state <= IDLE;
            done_reg <= 0;
            start <= 0;
            busy <= 0;
            baud_en <= 0;
            counter_halfperiod <= 2'b0;
            count_ones <= 0;
            count_zeros <= 0;
        end
        else begin
            case(state)
                IDLE: begin
                    done_reg <= 0;
                    baud_en <= 0;
                    if (rx_en == 1) begin
                        state <= START;
                        bit_in <= 0;
                        start <=1;
                    end
                end
```

```
                START: begin
                    if(d_in == 0) begin
                        baud_en <= 1;
                        if(counter_halfperiod < 4) begin
                            if(count_8x_ready == 1) begin
                                counter_halfperiod = counter_halfperiod + 1;
                            end
                        end
                        else begin
                            state <= DATA;
                            busy <=1;
                            start <= 0;
                            counter_halfperiod = 2'b0;
                            baud_en <= 0;
                        end
                    end
                end

                DATA: begin
                    baud_en <= 1;
                    if(count_baud_ready == 1) begin
                        if(count_ones > count_zeros) begin
                            d_out_reg = {1'b1,d_out_reg[7:1]};
                        end
                        if(count_ones < count_zeros) begin
                            d_out_reg = {1'b0,d_out_reg[7:1]};
                        end
                        if(bit_in == 7) begin
                            busy <= 0;
                            state <= STOP;
                            bit_in <= 0;
                        end
                        else begin
                            bit_in <= bit_in + 1;
                        end
                        count_ones = 3'b0;
                        count_zeros = 3'b0;
                    end

                    else begin
                        if(count_8x_ready == 1) begin
                            if(d_in == 1) begin
                                count_ones = count_ones + 1;
                            end

                            else begin
                                count_zeros = count_zeros + 1;
                            end
                        end
                    end
                end

                STOP: begin
                    if(count_baud_ready == 1) begin
                        state <= IDLE;
                        baud_en <= 0;
                        done_reg <= 1;
                    end
                end

                default: begin
                    state <= IDLE;
                    done_reg <= 0;
                    d_out_reg <= 1;
                    baud_en <= 0;
                end

            endcase
        end
    end
    assign done = done_reg;
    assign d_out = d_out_reg;
endmodule
```

Since I made the receiver before writing the top module, I had to instantiate the baud_gen module in the receiver. When I made the top module, I connected them all in the top module.

Receiver is normally in IDLE state. When rx_en gets logic 1, it switches to START state and the system starts to work.

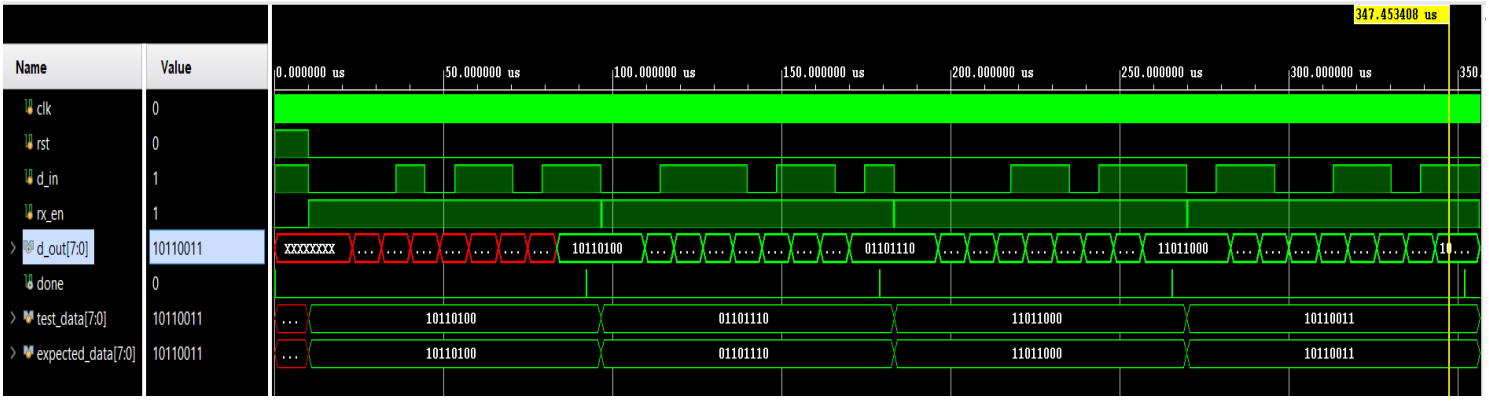
In START state, it waits for half of the baud period when it detects the zero (start) bit. I measured half of the baud period with 8x baud frequency 4 times. After waiting half of the baud period, it switches to DATA state.

In the DATA state, it samples from the d_in input every 8x frequency. When it receives 8 times (when the baud frequency is ready), it compares the number of ones with the number of zeros. Whichever one is more, it outputs it. When all values in the input are read (I checked with bit_in), it switches to STOP state.

In STOP state, it waits for the baud period. Then it gives the stop bit (logic 1) and returns to IDLE state.

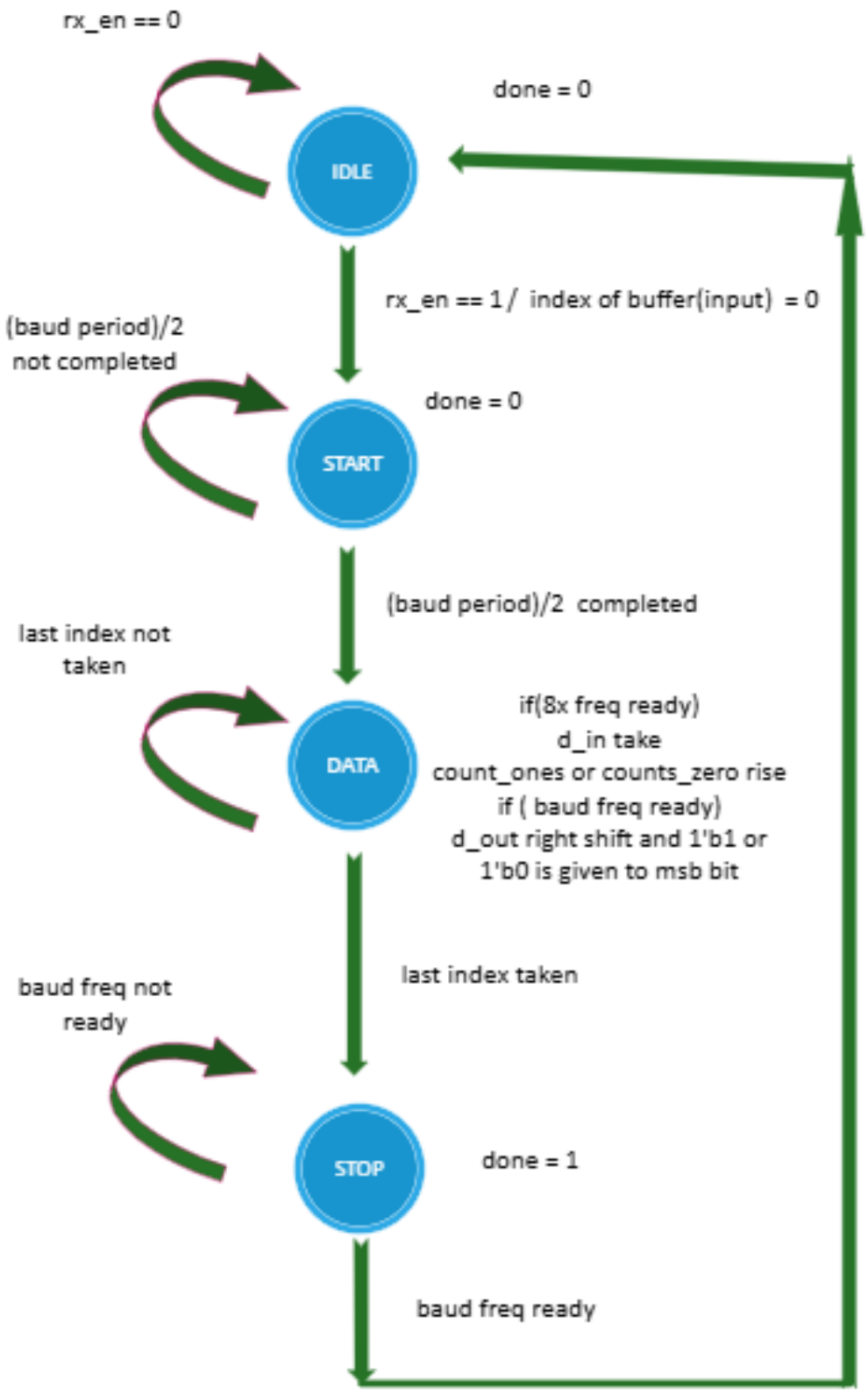
SIMULATION WAVEFORM

Because d_out has a value of X when the script starts running. It gives X value to the bits that are not filled until it is filled with the first 8 bits. Then it recovers. The top photo shows that it gives the result of the first data correctly. The value we give is the test_data value, we give this value in series with d_in and see it at the output in parallel with d_out.



I gave 4 different values in the top photo. The code correctly extracted them all in parallel.

FINITE STATE MACHINE



SIMULATION SOURCE

```
module uart_rx_tb;

    parameter freq = 115200; // Baud rate
    reg clk = 0;
    reg rst = 0;
    reg d_in = 1; // IDLE STATE
    reg rx_en = 0;
    wire [7:0] d_out;
    wire done;
    reg [7:0] test_data;
    reg [7:0] expected_data;
    integer file, i;
    // Clock Ã¼retimi (100 MHz clock, 10 ns period)

    always #5 clk = ~clk;

    // Test edilen UART RX modÃ¼lÃ¼
    uart_rx #(.freq(freq)) uut (
        .clk(clk),
        .rst(rst),
        .d_in(d_in),
        .rx_en(rx_en),
        .d_out(d_out),
        .done(done)
    );

    // UART veri gÃ¶nderim simÃ¼lasyonu
    task send_uart_byte(input [7:0] para_in);
        integer i;
        begin
            // Start bit (0)
            d_in = 0;
            rx_en = 1;
            #(8640);

            // Data bitleri (LSB'den MSB'ye)
            for (i = 0; i < 8; i = i + 1) begin
                d_in = para_in[i];
                #(8640);
            end

            // Stop bit (1)
            d_in = 1;
            #(8640);
        end
    endtask

    // Test senaryosu
    initial begin
        file = $fopen("stimulus2.txt", "r");
        if (file == 0) begin
            $display("Error: Cannot open stimulus2.txt");
            $finish;
          end

        $display("Starting UART RX testbench...");
        rst = 1;
        #10000;
        rst = 0;
        rx_en = 0;
        #45;
```

```
        i = 0;
        while (!$feof(file)) begin
            $fscanf(file, "%b\n", test_data);
            expected_data = test_data;

            rx_en = 1;  // RX etkinleştir
            send_uart_byte(test_data);

            #100;

            rx_en = 0;  // Yeni veri için devreyi sıfırla
            #60;

            if (d_out == expected_data) begin
                $display("Test %0d Passed! Sent: %b, Received: %b", i + 1, expected_data,
d_out);
            end else begin
                $display("Test %0d Failed! Sent: %b, Received: %b", i + 1, expected_data,
d_out);
            end

            i = i + 1; // Sayaç doğru artırılıyor
        end

        $display("Test completed!");
        $fclose(file);
        $finish;
    end

endmodule
```

I created a task to test the receiver in the simulation code. I tested it by taking 4 data from the file I created with the Phyton code. I compared the result I obtained with the result that should be and printed it to the console.

TOP MODULE

DESIGN SOURCE

```
module uart_top #(parameter [16:0] freq = 115200) (
    input clk,
    input rst,
    input [7:0] data_in_tx,
    input tx_en,
    input rx_en,
    output [7:0] data_out_rx,
    output tx_done,
    output rx_done,
    output tx_start,
    output rx_start,
    output tx_busy,
    output rx_busy,
    output tx_out
);

    wire tx_baud_en,tx_count_baud_ready;
    wire rx_baud_en,rx_count_baud_ready,rx_count_8x_ready;


    // RX için baud gen modülü
    baud_gen #(.freq(freq)) rx_baud_gen (
        .clk(clk),
        .rst(rst),
        .baud_en(rx_baud_en),
        .count_8x_ready(rx_count_8x_ready),
        .count_baud_ready(rx_count_baud_ready)
    );

    // TX için baud gen modülü
    baud_gen #(.freq(freq)) tx_baud_gen (
        .clk(clk),
        .rst(rst),
        .baud_en(tx_baud_en),
        .count_8x_ready(),
        .count_baud_ready(tx_count_baud_ready)
    );

    // UART alıcı modülü (RX)
    uart_rx #(.freq(freq)) rx_inst (
        .clk(clk),
        .rst(rst),
        .d_in(tx_out),
        .rx_en(rx_en),
        .baud_en(rx_baud_en),
        .count_8x_ready(rx_count_8x_ready),
        .start(rx_start),
        .busy(rx_busy),
        .count_baud_ready(rx_count_baud_ready),
        .d_out(data_out_rx),
        .done(rx_done)
    );

    // UART verici modülü (TX)
    uart_tx #(.freq(freq)) tx_inst (
        .clk(clk),
        .rst(rst),
        .d_in(data_in_tx),
        .tx_en(tx_en),
        .count_baud_ready(tx_count_baud_ready),
        .start(tx_start),
        .busy(tx_busy),
        .baud_en(tx_baud_en),
        .seri_out(tx_out),    // Seri çıkış
        .done(tx_done)        // Verici işlem tamamlandı sinyali
    );

endmodule
```

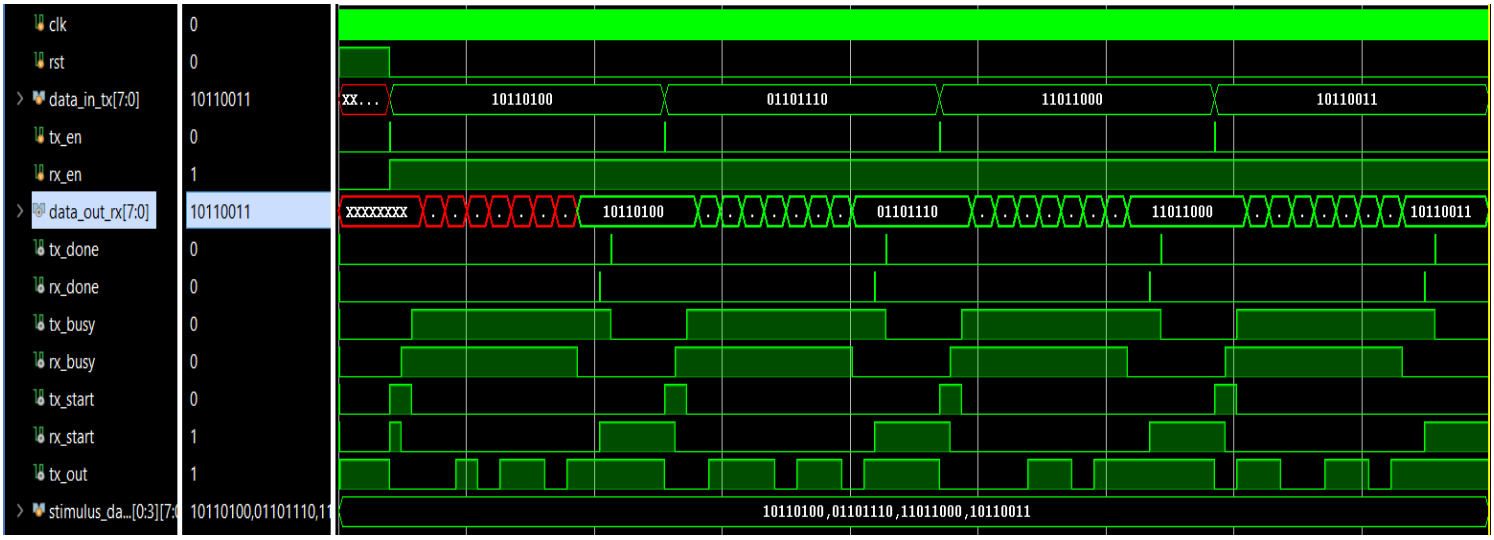
I instantiate a transmitter, a receiver and two baud_gen. A baud_gen for receiver other for transmitter. I made all connections in top module. In the top module, I receive 8-bit parallel data and give it to the input of the transmitter. I give the output of the transmitter to the input of the receiver. I give the output of the receiver to the output of the top module.

SIMULATION SOURCE

I give the file I created in Phyton to the transmitter input. Finally, I compare the data_out I get from the receiver with the data from the file.

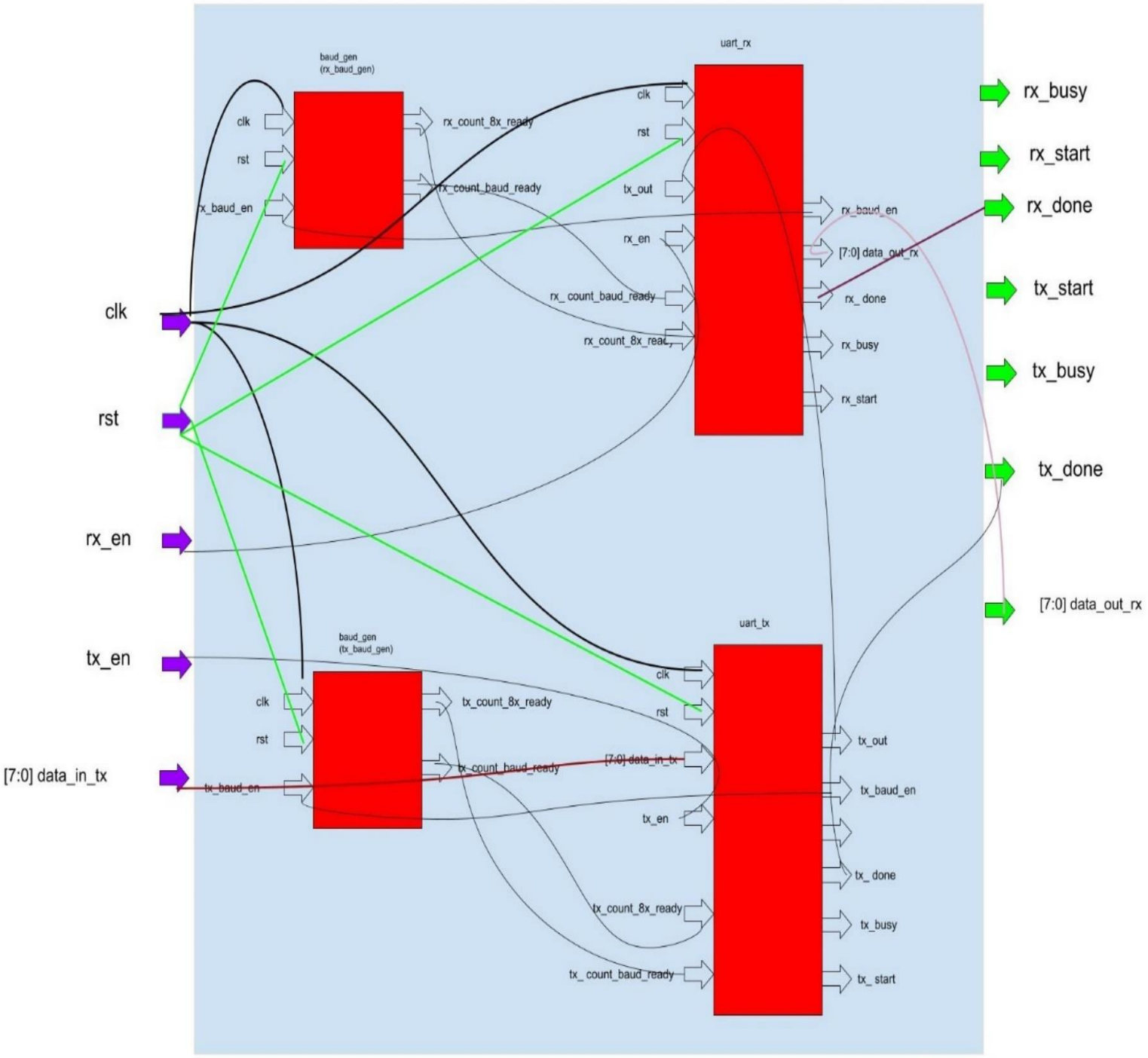
```
module uart_top_tb();
    reg clk = 1'b0;
    reg rst = 1'b0;
    reg [7:0] data_in_tx;
    reg tx_en = 0;
    reg rx_en = 0;
    wire [7:0] data_out_rx;
    wire tx_done;
    wire rx_done;
    wire tx_busy;
    wire rx_busy;
    wire tx_start;
    wire rx_start;
    wire tx_out;
    integer  i;
    // UART top modül örneği
    uart_top uut (
        .clk(clk),
        .rst(rst),
        .data_in_tx(data_in_tx),
        .tx_en(tx_en),
        .rx_en(rx_en),
        .data_out_rx(data_out_rx),
        .tx_done(tx_done),
        .rx_done(rx_done),
        .tx_busy(tx_busy),
        .rx_busy(rx_busy),
        .tx_start(tx_start),
        .rx_start(rx_start),
        .tx_out(tx_out)
    );
    integer file;
    reg [7:0] stimulus_data[0:3];

    // Saat sinyali üretimi
    always #5 clk = ~clk;  // 10ns periyotlu (100 MHz) saat

    initial begin

        file = $fopen("stimulus2.txt", "r");
        if (file == 0) begin
            $display("Error: Cannot open stimulus2.txt");
            $finish;
        end


        for (i = 0; i < 4; i = i + 1) begin
            $fscanf(file, "%b\n", stimulus_data[i]);
        end

        $fclose(file);

        rst = 1;
        #20000;
        rst = 0;
        tx_en = 0;
        rx_en = 0;

        for (i = 0; i < 4; i = i + 1) begin
            data_in_tx = stimulus_data[i];
            tx_en = 1;
            #10;
            tx_en = 0;
            rx_en = 1;
            wait(tx_done);
            #1000;
            if (data_out_rx == stimulus_data[i]) begin
                $display("Data received correctly: %h", data_out_rx);
            end else begin
                $display("Error: Received data %h, expected %h",
data_out_rx, stimulus_data[i]);
            end
            #20000;
           end

        $finish;  // Test tamamlandı
    end
endmodule
```

SIMULATION WAVEFORM



I receive the data from the file with data_in_tx and receive it from the transmitter. I give tx_out to the output in serial. At the same time, I give the tx_out output of the transmitter to the input of the receiver. Finally, the data coming in with data_in_tx comes out of the receiver correctly as data_out_tx.

BLOCK DIAGRAM



UTILIZATION REPORTS

Utilization Post-Synthesis | **Post-Implementation**

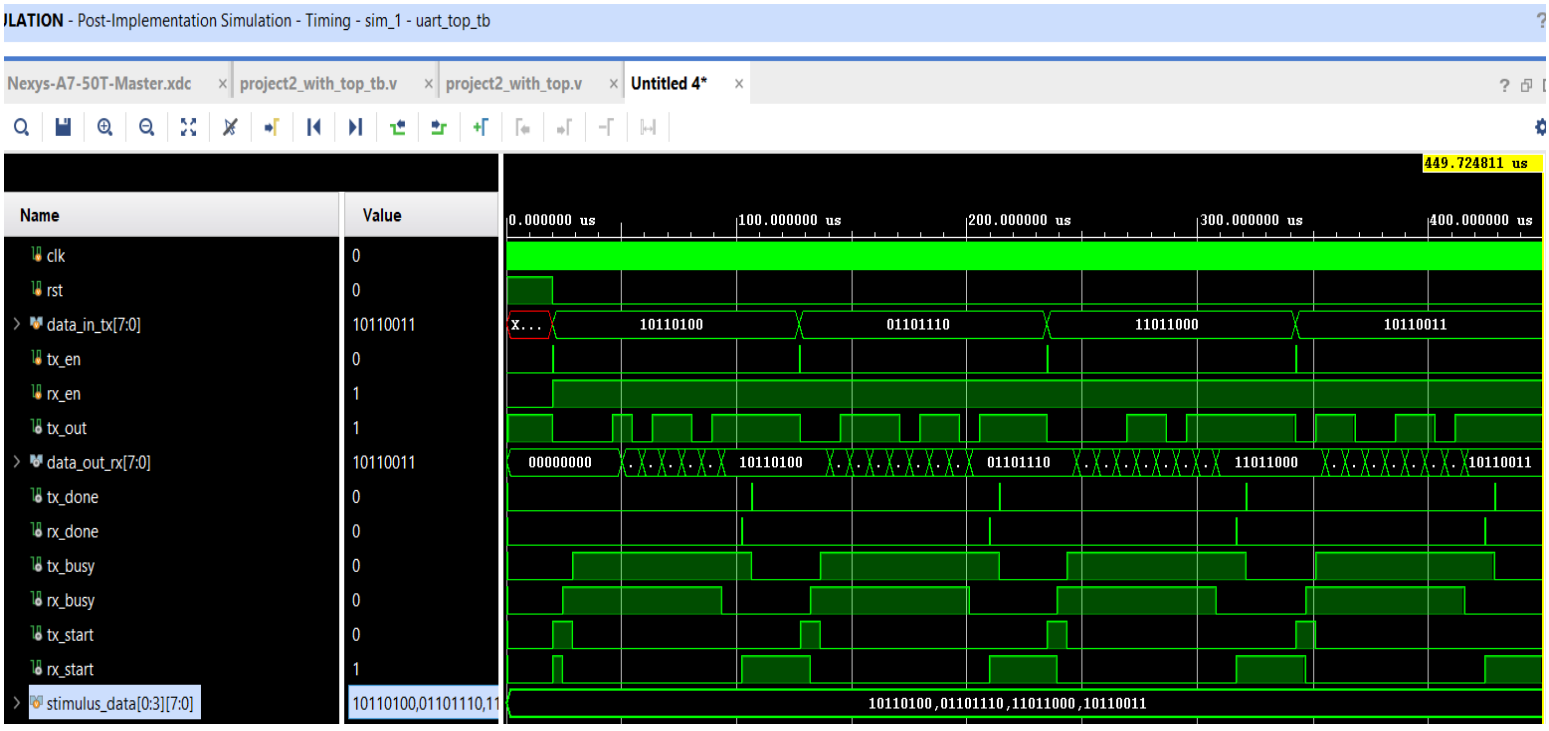
Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	78	32600	0.24
FF	91	65200	0.14
IO	27	210	12.86
BUFG	1	32	3.13

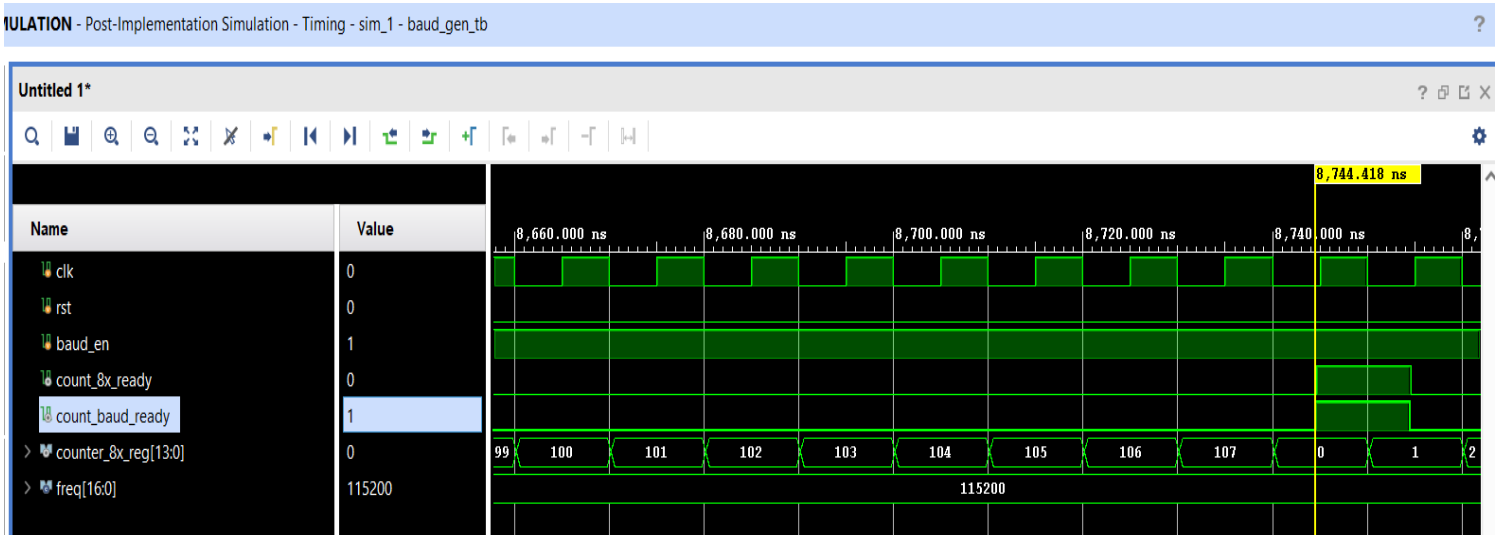
Our WHS value is 5.839 ns. Therefore, 10 – 5.839 = 4.161 ns. Maximum clk frequency is

1/ 4.161 ns = 240 MHz

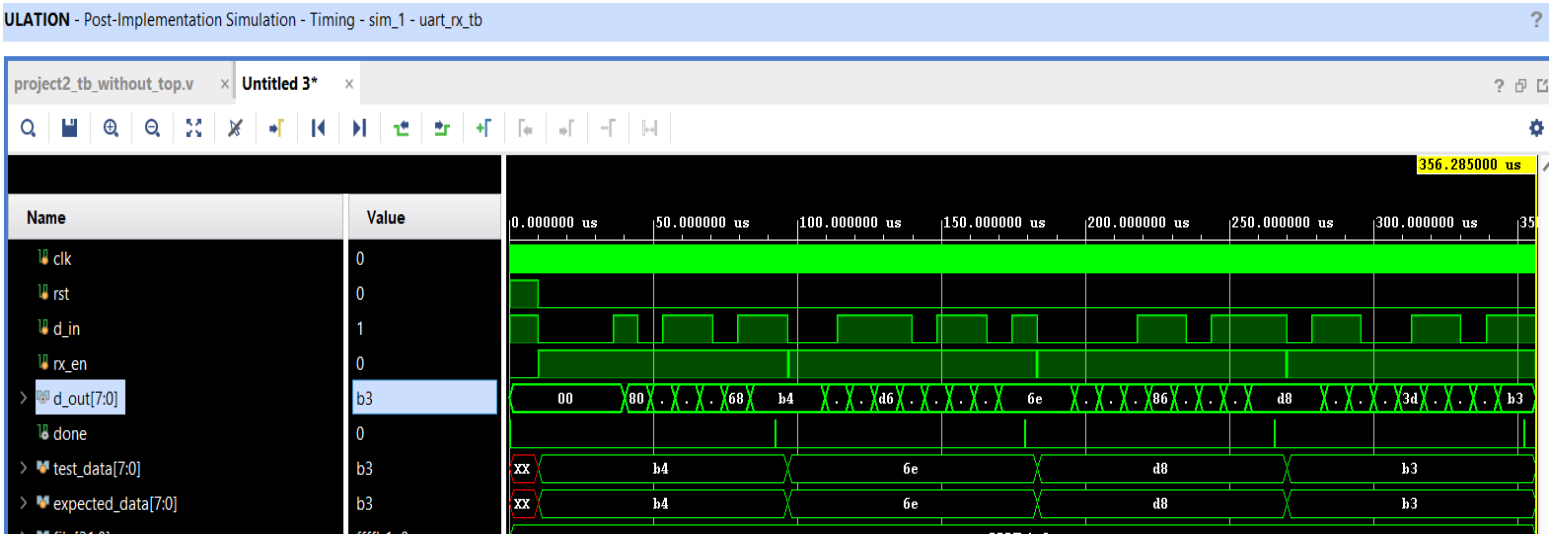
POST – TIMING IMPLEMENTATION SIMULATION



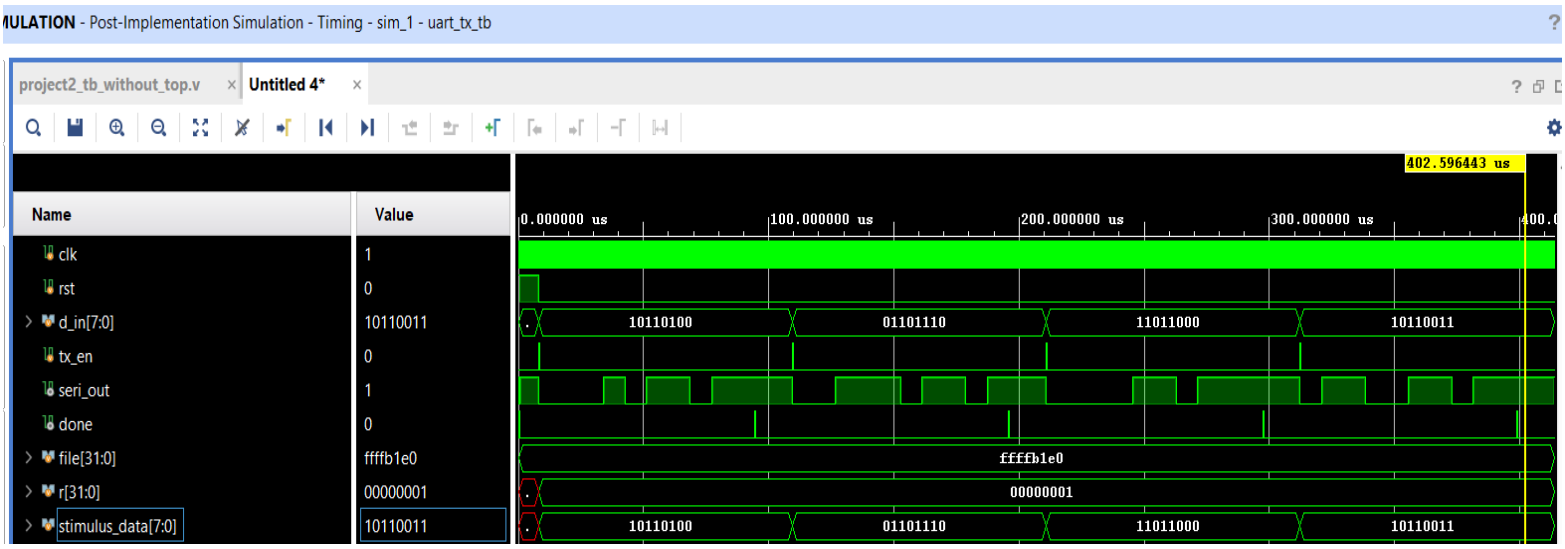
Top Module Simulation



Baud Generator Simulation



Receiver Simulation



Transmitter Simulation

REFERENCES

[1] Peňa, E., & Legaspi, M. G. (n.d.). *UART: A Hardware Communication Protocol Understanding Universal Asynchronous Receiver/Transmitter*. Analog Devices. https://www.analog.com/en/resources/analog-dialogue/articles/uart-a-hardware-communication-protocol.html

[2] Kelvin. (2022, December 15). *UART Communication Protocol and how it works - latest open tech from SEEED*. Latest Open Tech From Seeed. https://www.seeedstudio.com/blog/2022/09/08/uart-communication-protocol-and-how-it-works/

[3] *01signal: Vivado: Finding the "maximal frequency" after synthesis*. (n.d.). https://www.01signal.com/vendor-specific/xilinx/vivado-minimal-period/

[4] [Mehmet Burak Aykenar]. (2021, January 11). *VHDL ile FPGA PROGRAMLAMA - Ders18: UART Receiver Tasarımı ve Simülasyonu - Sıfırdan Kod Yazma* [Video]. Youtube. https://www.youtube.com/watch?v=d7XPkZDWWQY&t=1052s

[5] [[Mehmet Burak Aykenar]. (2021, January 8). *VHDL ile FPGA PROGRAMLAMA - Ders16: VHDL UART Transmitter Sıfırdan Kod Yazma* [Video]. YouTube. https://www.youtube.com/watch?v=5MzjKMZfLTk&t=1904s

[6] [Muhammed Kocaoğlu]. (2023, April 2). *SystemVerilog - UART Transmitter* [Video]. Youtube. https://www.youtube.com/watch?v=bh_RxlzEnvU&t=757s

[7] [Muhammed Kocaoğlu]. (2023, April 14). *SystemVerilog - UART Receiver* [Video]. Youtube. https://www.youtube.com/watch?v=va-q8QKqeMg&t=596s