آزمایش هفتم: UART

UART

پروتوکل یوآرت برای این آزمایش را به صورت روبرو در نظر گرفتیم. ابتدا باید داده ای را که می‌خواهیم ارسال کنیم را روی data_in قرار دهیم، سپس in_ready را برابر یک قرار دهیم تا sender از حالت IDLE خارج شده و ارسال را شروع کند. همانطور که در صورت آزمایش آمده یک بیت START_BIT سپس یک بیت PARITY سپس ۷ بیت داده و در نهایت هم یک بیت STOP ارسال می‌شود. در یوآرت مقیاسی برای هماهنگی دو ماژول فرستنده و گیرنده بنام buad rate که نشان می‌دهد در هر ثانیه چند بیت ارسال می‌شود. ما از پارامتری بنام CLKS_PER_BIT استفاده کردیم که نشان می‌دهد جابهجایی یک بیت چند کلاک طول می‌کشد و این عدد از تقسیم سرعت کلاک پردازنده بر buad rate بدست می‌آید. ما برای سهولت کار در سیمولیشن CLKS_PER_BIT را برابر ۲ در نظرگرفتیم و در کوارتوس برابر ۴۳۴ (با فرض buad rate =115200 و clk rate = 50MHz). (لینک روبرو نیز درفهم پروتوکول یوآرت مفید بود https://www.circuitbasics.com/basics-uart-communication/)

کارکرد ماژول فرستنده همانند توضحات آمده در صورت آزمایش است و هر دو کلاک (CLK_PER_BIT) کار می‌کند.

در بخش گیرنده، گیرنده در ابتدا در حالت IDLE است و پس از مشاهده این که بیت RX صفر شد وارد حالت START_BIT می‌شود و به مدت CLK_PER_BIT-1 بار (یک کلاک کمتر از حالت معمول) چک می‌کند که آیا بیت شروع همچنان صفر است یا خیر. در صورتی که در این بین خطایی رخ دهد گیرنده دوباره به حالت IDLE بازمی‌گردد.

سپس در حالت های بیت زوجیت و دریافت داده بعد از گذشتن دو کلاک داده مربوطه را می‌گیرد. در نهایت هم در حالت STOP دو کلاک صبر می‌کند و out_valid را به مدت یک کلاک برابر یک می‌کند (به معنای اینکه دریافت داده تمام شد) و به حالت IDLE می‌رود و منتظر می‌ماند تا دوباره RX از ولتاژ یک به صفر بیاید. توجه کنید که در حالت پایانی فرستنده TX باید برابر یک باشد که به معنی پایان کار آن باشد تا هر وقت خواست بتواند با صفر کردن آن به ماژول گیرنده بفهماند که ارسال داده شروع شده.

گیرنده داده هایی را که دریافت می‌کند به طور موقتی در رجیستری بنام current_data می‌ریزد و هر وقت که کار ارسال به پایان رسید آن را به خروجی ماژول گیرنده انتقال می‌دهد که بیت کم ارزش این خروجی زوجیت و ۷ بیت پرارزش بعدی داده‌ی دریافت شده است.

توجه کنید که برای سنتز ماژول یوآرت چون تستبنچ را سنتز نمی‌کنیم، مجبوریم تا پارامتر CLKS_PER_BIT را بصورت دستی با از کامنت در آوردن خط ۱۴ ماژول UART و حذف پارامتر از تعریف ماژول آن، ماژول قابلیت سنتز پیدا کند.

کد مربوط به uart :

```
module  uart #(parameter CLKS_PER_BIT) (data_in , data_out , clk, in_ready, out_valid
);

    input [6:0] data_in;
    input clk;
    input in_ready;

    output [7:0] data_out;
    output out_valid;

    //=====================================

    //fpga clock is at least about 50MHz
    //baud rate is normaly set at 115200
    // so CLKS_PER_BIT will be 50000000/115200 = 434
    //  parameter CLKS_PER_BIT = 434;

    //=====================================

    wire tx_2_rx;

    //=====================================

    sender  #(.CLKS_PER_BIT(CLKS_PER_BIT))    sender( .clk(clk),
                                                      .data_in(data_in),
                                                      .in_ready(in_ready),
                                                      .tx(tx_2_rx)
                                                      );

    reciver #(.CLKS_PER_BIT(CLKS_PER_BIT))    reciver(.clk(clk),
                                                      .rx(tx_2_rx),
                                                      .data_out(data_out),
                                                      .out_valid(out_valid) //just dete
rmine by stop-bit
                                                      );


endmodule
```

کد مربوط به بخش reciver:

```
`define DATA_LEN 7

module reciver #(parameter CLKS_PER_BIT)(clk, rx, data_out, out_valid);
```

```
    input clk;
    input rx;

    output reg [`DATA_LEN:0] data_out;
    output reg out_valid = 0;

    //====================================

    reg [2:0] state = 3'b000, next_state;
    reg [`DATA_LEN:0] current_data = 0;
    reg par;
    reg [3:0] data_index = 0;
    reg clk_count = 0;

    //====================================

    parameter IDLE          = 3'b000;
    parameter START_BIT     = 3'b001;
    parameter PARITY        = 3'b010;
    parameter RECIVE_DATA   = 3'b011;
    parameter STOP          = 3'b100;

    //====================================

    always @(*)begin
        state <= next_state;
    end

    reg [10:0] cc =0;

    always@(posedge clk)begin
        cc = cc + 1;
        case (state)
            IDLE: begin
                out_valid = 0;
                data_index = 0;
                clk_count = 0;
                if(rx == 1'b0)begin
                    next_state = START_BIT;
                end
                else begin
                    next_state = IDLE;
                end
            end

            START_BIT:begin
                if(clk_count < CLKS_PER_BIT-2 )begin
                    clk_count = clk_count + 1;
                    next_state = START_BIT;
```

```
            if(rx == 1'b1)begin
                clk_count =0;
                next_state = IDLE;
            end
        end
        else begin
            clk_count = 0;
            next_state = PARITY;
        end
    end

    PARITY: begin
        if(clk_count < CLKS_PER_BIT-1)begin
            clk_count = clk_count +1;
            next_state = PARITY;
        end
        else begin
            clk_count = 0;
            next_state = RECIVE_DATA;
            current_data[0] = rx;
        end

    end

    RECIVE_DATA: begin
        if (clk_count < CLKS_PER_BIT-1)begin
            clk_count = clk_count + 1;
            next_state = RECIVE_DATA;
        end
        else begin
            clk_count = 0;
            current_data[data_index + 1] = rx;
            if(data_index < `DATA_LEN-1) begin
                data_index = data_index + 1;
                next_state = RECIVE_DATA;
            end
            else begin
                data_index = 0;
                next_state = STOP;
            end
        end
    end

    STOP: begin
        if(clk_count < CLKS_PER_BIT-1) begin
            clk_count = clk_count + 1;
            next_state = STOP;
        end
        else begin
```

```
                    clk_count = 0;
                    next_state = IDLE;
                    out_valid = 1'b1;
                    data_out = current_data;
                end
            end

            default: begin
                next_state =IDLE;
            end
        endcase
    end

endmodule
```

کد مربوط به بخش sender:

```
`define DATA_LEN 7

module sender #(parameter CLKS_PER_BIT) (clk, data_in, in_ready, tx);

    input clk;
    input [`DATA_LEN-1:0] data_in;
    input in_ready;

    output tx;

    //=====================================

    reg tx;
    reg [2:0] state = 3'b000, next_state;
    reg [3:0] data_index = 0;
    reg [7:0] clk_count = 0;
    reg [`DATA_LEN-1:0] current_data = 0;

    wire par;

    //=====================================

    parameter IDLE          = 3'b000;
    parameter PARITY        = 3'b001;
    parameter START_BIT     = 3'b010;
    parameter TRANSFER_DATA = 3'b011;
    parameter STOP_BIT      = 3'b100;

    //=====================================
```

```
    assign par =    current_data[0] ^ current_data[1] ^ current_data[2] ^ current_dat
a[3] ^
                    current_data[4] ^ current_data[5] ^ current_data[6];

    always @(*)begin
        state <= next_state;
    end

    reg [10:0] cc = 0;

    always @(posedge clk) begin
        next_state = IDLE;
        cc = cc +1;
        case (state)
            IDLE:begin
                clk_count = 0;
                data_index = 0;
                tx = 1'b1;
                if (in_ready == 1'b1) begin
                    current_data = data_in;
                    next_state = START_BIT;
                end
                else begin
                    next_state = IDLE;
                end
            end

            START_BIT: begin
                tx = 1'b0;
                if(clk_count < CLKS_PER_BIT-1) begin
                    clk_count = clk_count + 1;
                    next_state = START_BIT;
                end
                else begin
                    clk_count = 0;
                    next_state = PARITY;
                end
            end

            PARITY: begin
                tx = par;
                if(clk_count < CLKS_PER_BIT-1)begin
                    clk_count = clk_count + 1;
                    next_state = PARITY;
                end
                else begin
                    clk_count =0;
                    next_state = TRANSFER_DATA;
                end
```

```
            end

            TRANSFER_DATA:begin
                tx = current_data[data_index];
                if(clk_count < CLKS_PER_BIT-1)begin
                    clk_count = clk_count + 1;
                    next_state = TRANSFER_DATA;
                end
                else begin
                    clk_count = 0;
                    if (data_index < `DATA_LEN - 1)begin
                        data_index = data_index + 1;
                        next_state = TRANSFER_DATA;
                    end
                    else begin
                        data_index = 0;
                        next_state = STOP_BIT;
                    end
                end
            end

            STOP_BIT:begin
                tx = 1'b1;
                if(clk_count < CLKS_PER_BIT-1)begin
                    clk_count = clk_count + 1;
                    next_state = STOP_BIT;
                end
                else begin
                    clk_count = 0;
                    next_state = IDLE;
                end
            end

            default: begin
                next_state = IDLE;
            end
        endcase
    end

endmodule
```

کد مربوط به تست بنچ:

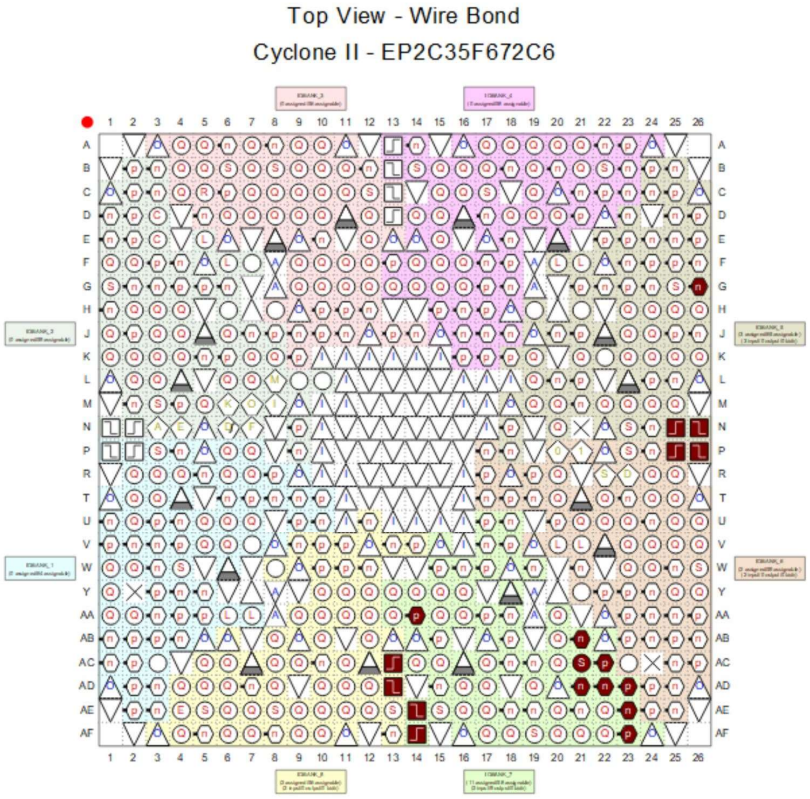
```
module test_uart();

    parameter clks_per_bit = 2;

    //=====================================
```

```
    reg [6:0] data_in;
    reg clk;
    reg in_ready;

    wire [7:0] data_out;
    wire out_valid;

    //=====================================

    UART  #(.CLKS_PER_BIT(clks_per_bit)) uart (.data_in(data_in),
                                               .data_out(data_out),
                                               .clk(clk),
                                               .in_ready(in_ready),
                                               .out_valid(out_valid)
                                               );

    //=====================================

    initial begin
        clk = 0;
        in_ready = 0;
        #12
        in_ready = 1;
        data_in = 7'b1101101;
        #15
        in_ready = 0;
        ////////////////////////////////////////////
        #220
        in_ready = 1;
        data_in = 7'b0101010;
        #15
        in_ready = 0;
    end

    always #5 clk = ~clk;

endmodule
```

در تست بنچ بالا یکبار داده ۱۱۰۱۱۰ و یکبار داده ۰۱۰۱۰۱۰ را با فاصله ارسال کردیم و در بخش پایانی ویوفرم آن و توضیحات مربوطه آمده است.

اختصاص پین و کامپایل

برای پیاده‌سازی این مدار از دستگاه EP2C35F672C6 از خانواده Cyclone II استفاده خواهد شد. با توجه به راهنمای این دستگاه، پین‌ها را مطابق شکل 1 اختصاص می‌دهیم.



clk	Input	PIN_P26	6	B6_N0	PIN_P26	3.3-V LV...default)	24mA (default)
data_in[6]	Input	PIN_N25	5	B5_N1	PIN_N25	3.3-V LV...default)	24mA (default)
data_in[5]	Input	PIN_N26	5	B5_N1	PIN_N26	3.3-V LV...default)	24mA (default)
data_in[4]	Input	PIN_P25	6	B6_N0	PIN_P25	3.3-V LV...default)	24mA (default)
data_in[3]	Input	PIN_AE14	7	B7_N1	PIN_AE14	3.3-V LV...default)	24mA (default)
data_in[2]	Input	PIN_AF14	7	B7_N1	PIN_AF14	3.3-V LV...default)	24mA (default)
data_in[1]	Input	PIN_AD13	8	B8_N0	PIN_AD13	3.3-V LV...default)	24mA (default)
data_in[0]	Input	PIN_AC13	8	B8_N0	PIN_AC13	3.3-V LV...default)	24mA (default)
data_out[7]	Output	PIN_AA14	7	B7_N1	PIN_AA14	3.3-V LV...default)	24mA (default)
data_out[6]	Output	PIN_AC21	7	B7_N0	PIN_AC21	3.3-V LV...default)	24mA (default)
data_out[5]	Output	PIN_AD21	7	B7_N0	PIN_AD21	3.3-V LV...default)	24mA (default)
data_out[4]	Output	PIN_AD23	7	B7_N0	PIN_AD23	3.3-V LV...default)	24mA (default)
data_out[3]	Output	PIN_AD22	7	B7_N0	PIN_AD22	3.3-V LV...default)	24mA (default)
data_out[2]	Output	PIN_AC22	7	B7_N0	PIN_AC22	3.3-V LV...default)	24mA (default)
data_out[1]	Output	PIN_AB21	7	B7_N0	PIN_AB21	3.3-V LV...default)	24mA (default)
data_out[0]	Output	PIN_AF23	7	B7_N0	PIN_AF23	3.3-V LV...default)	24mA (default)
in_ready	Input	PIN_G26	5	B5_N0	PIN_G26	3.3-V LV...default)	24mA (default)
out_valid	Output	PIN_AE23	7	B7_N0	PIN_AE23	3.3-V LV...default)	24mA (default)

شکل 1 - نحوه‌ی اختصاص پین‌ها

تنها برای سیگنال in_ready از Push-button استفاده کردیم و مابقی ورودی ها به غیر از کلاک را به Switch متصل کردیم و خروجی ها را نیز به LED و کلاک را نیز به external clock وصل کردیم.

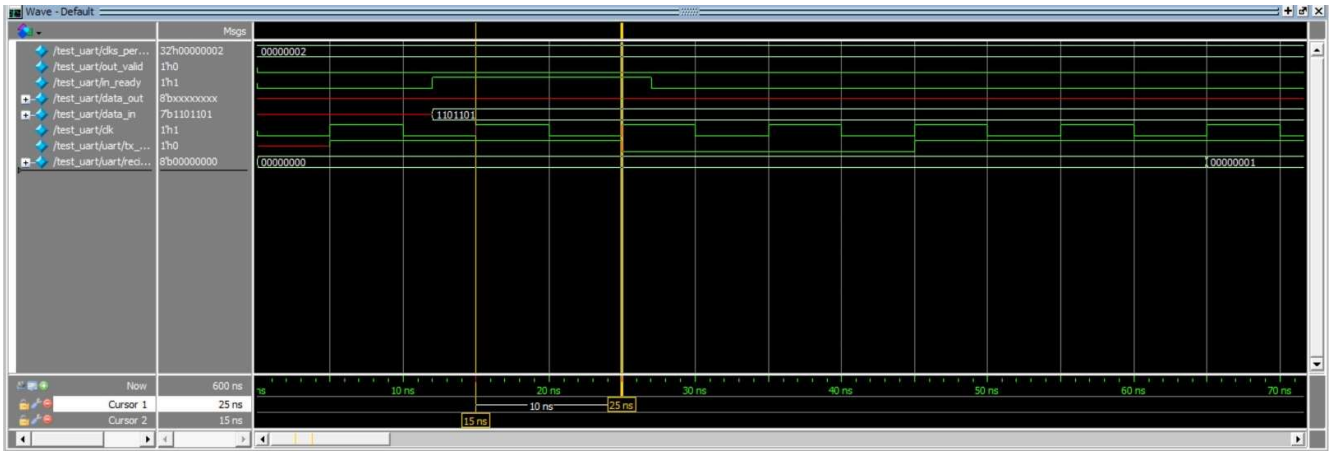
حال، پروژه را کامپایل می‌کنیم. نتیجه‌ی تحلیل و سنتز پس از کامپایل پروژه در شکل 2 نشان داده شده است.

Flow Summary	
Flow Status	Successful - Sun Jul 19 20:51:10 2020
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	uart
Top-level Entity Name	uart
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	0 / 33,216 (0 %)
Total combinational functions	0 / 33,216 (0 %)
Dedicated logic registers	0 / 33,216 (0 %)
Total registers	0
Total pins	18 / 475 (4 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

Task	Time
Compile Design	00:00:19
Analysis & Synthesis	00:00:03
Fitter (Place & Route)	00:00:08
Assembler (Generate programming files)	00:00:05
TimeQuest Timing Analysis	00:00:03
EDA Netlist Writer	
Program Device (Open Programmer)	

شکل 2 - نتیجه‌ی سنتز ماژول یوآرت

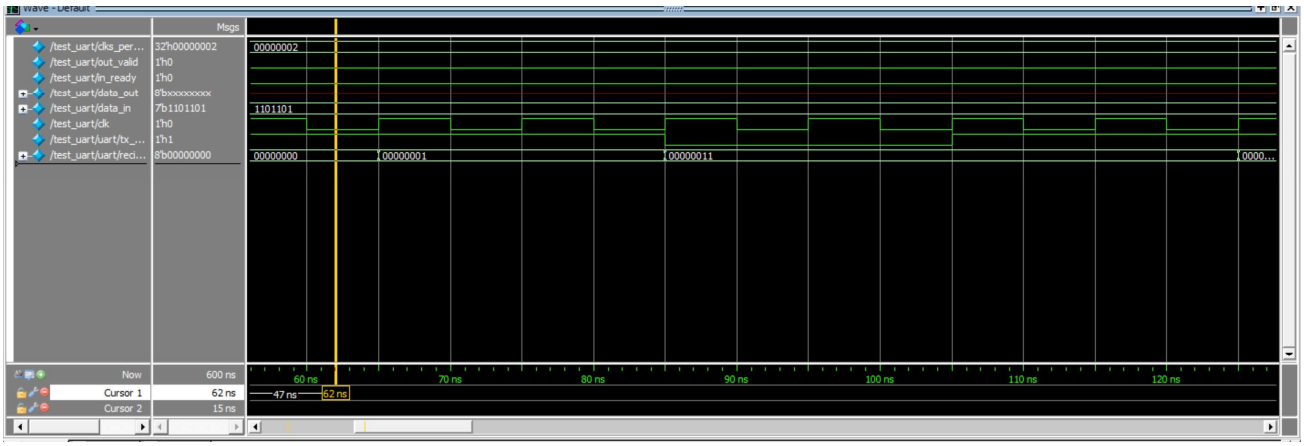
شکل 3، تست عملکرد ماژول پیاده‌سازی شده به کمک Waveform را نشان می‌دهد.

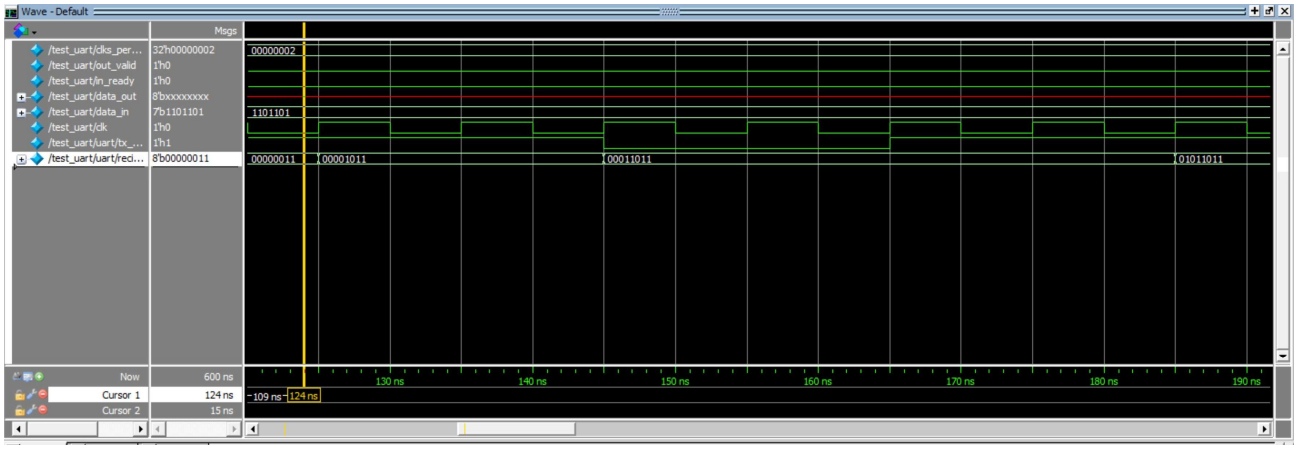


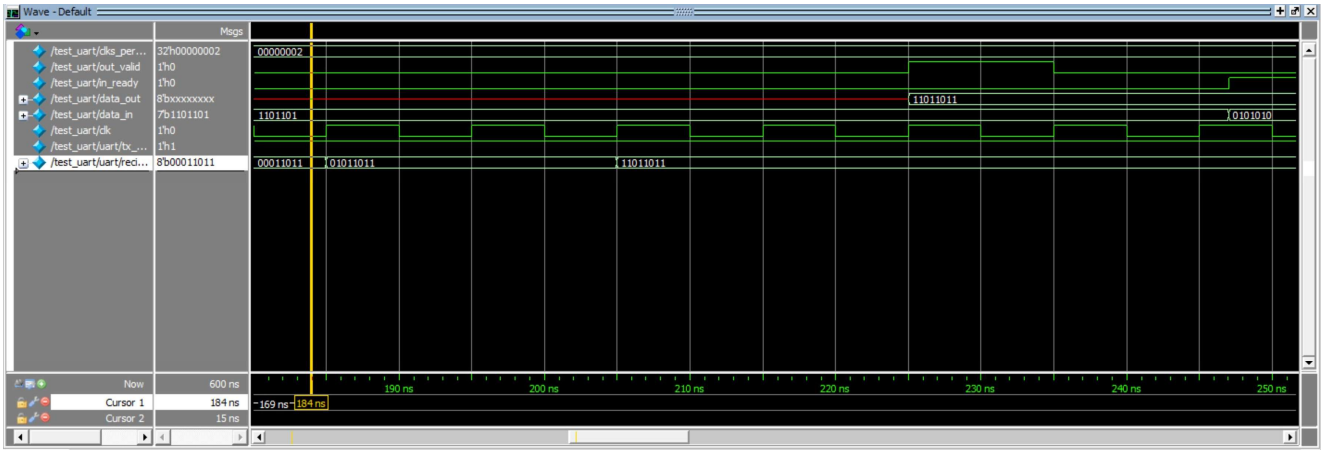
شکل 3 - شکل موج حاصل از تست عملکرد ماژول تست بنچ ضرب کننده بوث

در شکل بالا سیگنال ها را مشاهده می‌کنید. پایین ترین سیگنال مربوط به current_data از ماژول reciver است. دومین سیگنال از پایین نیز برابر سیم بین reciver وsender است. ما کمی زود تر داده را قرار می دهیم و سیگنال in_ready را یک می‌کنیم. درشکل

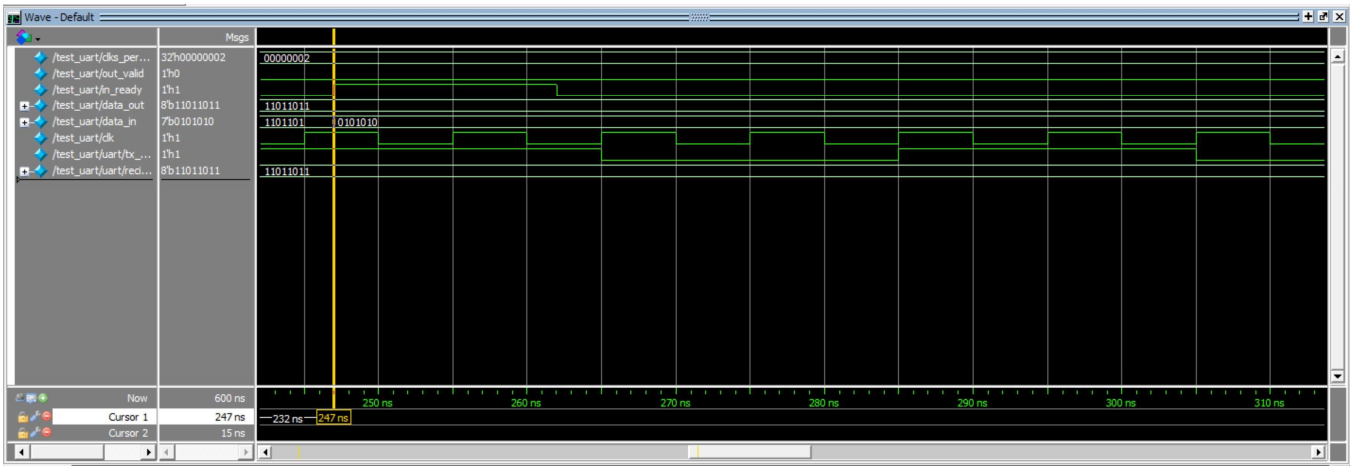
بالا بین دو نشانگر زرد sender در حالت IDLE است. پس از آن مشاهده می‌کنید که دو کلاک برابر ۰ است که به معنای شروع ارسال است. سپس داده های مربوطه را به مدت هر دو کلاک ارسال می‌کند که در شکل های زیر می‌توانید تغییر current_data را پس از پایان هر دو کلاک مشاهده کنید. (برای اینکه بتوان راحتر شکل ها را دنبال کرد یکی از نشانگر های زرد را جلو بردیم و در ابتدای هر در شکل قرار دادیم.)

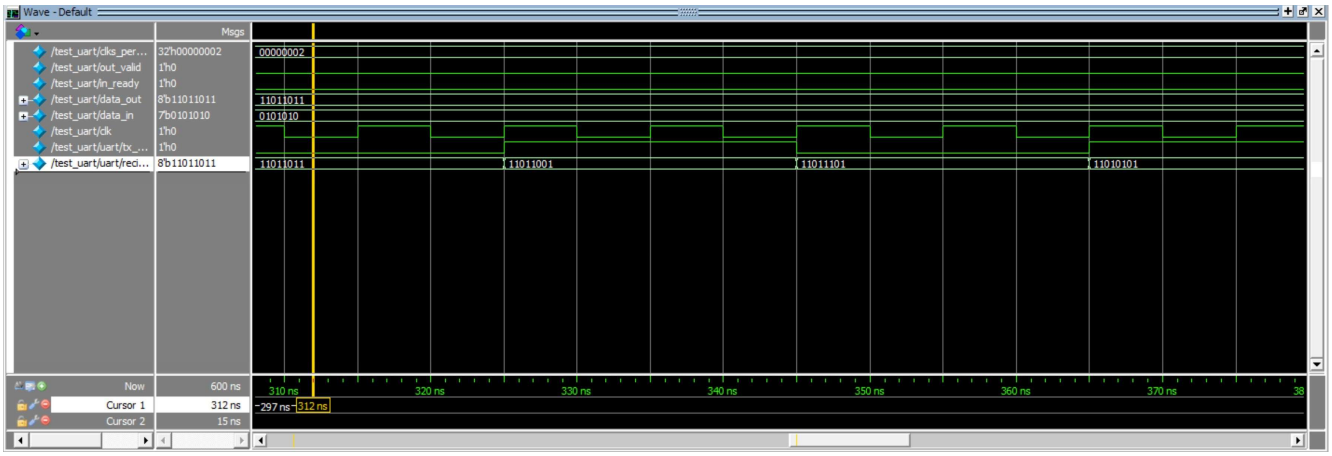


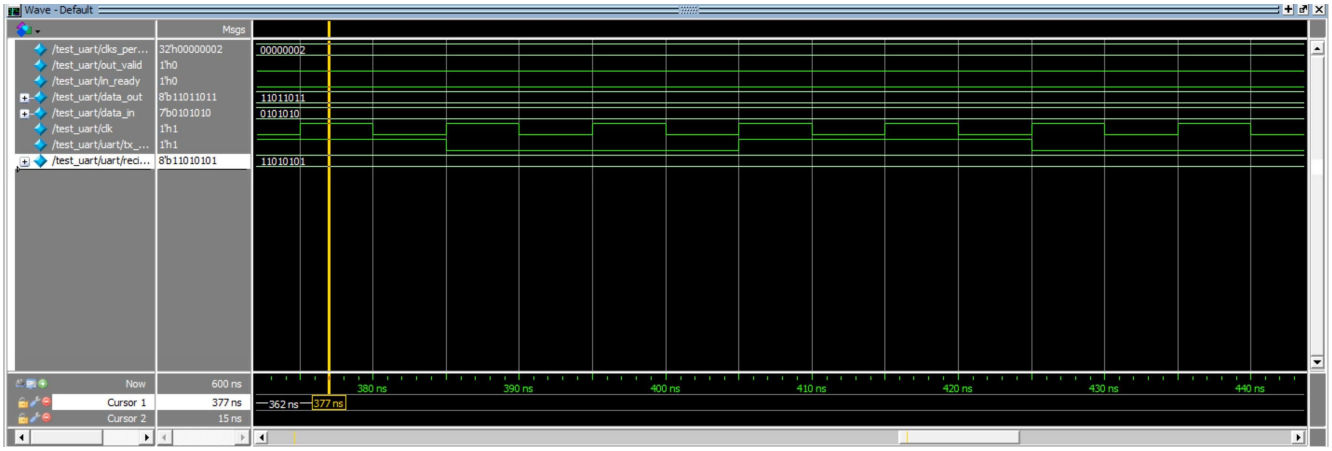


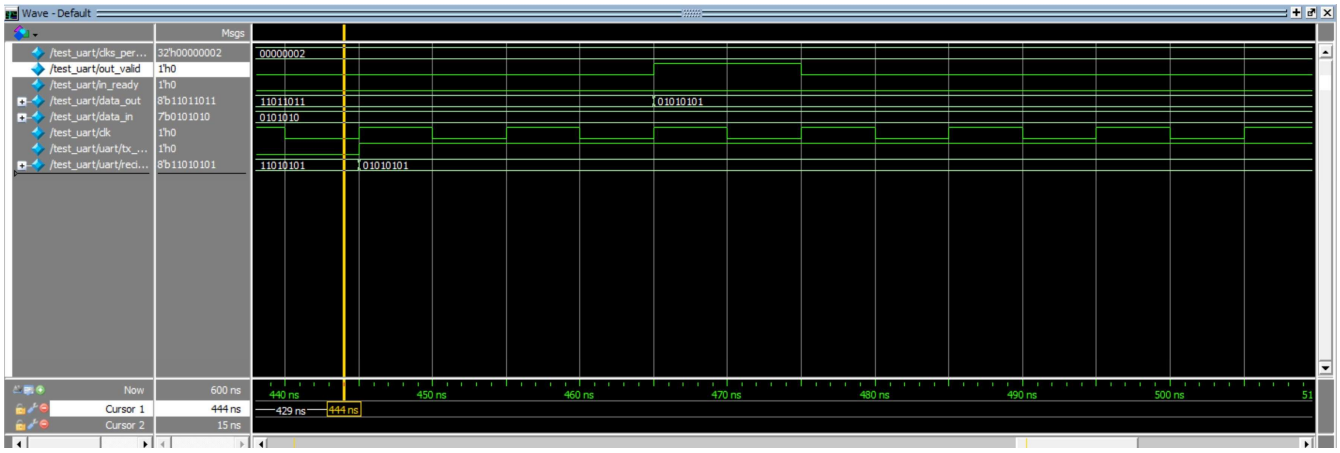


همانطور که در شکل بالا مشاهده می‌کنید دو کلاک پس از اینکه مقدار current_data به مقداری که ارسال شده بود تغییر کرد، خروجی یوآرت نیز عوض می‌شود و یک کلاک سیگنال out_ready را برابر یک می‌کند.(در واقع نشان می‌دهد حالت stop به پایان رسیده است). در شکل های زیر نیز ارسال داده‌ی بعدی را مشاهده می‌کنید. چون این داده ۰۱۰۱۰۱۰ است و بیت های آن یک در میان عوض می‌شود دیدن تغییرات current_data کمی راحتر است.








Wave - Default
Msgs
/test_uart/clks_per...
/test_uart/out_valid
/test_uart/in_ready
/test_uart/data_out
/test_uart/data_in
/test_uart/clk
/test_uart/uart/tx_...
/test_uart/uart/reci...
32'h00000002
1'h0
1'h0
8'b11011011
7'b0101010
1'h0
1'h0
8'b11010101
00000002
11011011
01010101
0101010
11010101
01010101
Now
600 ns
Cursor 1
444 ns
Cursor 2
15 ns
440 ns
450 ns
460 ns
470 ns
480 ns
490 ns
500 ns
429 ns
444 ns