

گزارش آزمایش هفتم



دانشگاه صنعتی شریف - بهار ۱۴۰۰

آزمایشگاه طراحی سیستم‌های دیجیتال - دکتر اجلالی

نویسندگان:

سروش جهانزاد - ۹۸۱۰۰۳۸۹

علی حاتمی تاجیک - ۹۸۱۰۱۳۸۵

۱ | مقدمه

در این گزارش روند طراحی یک ارسال کننده سریال UART آورده شده است. سعی شده تا تمام موارد به صورت کامل و کافی توضیح داده شود. عکس‌های نتایج شبیه‌سازی در فولدر گزارش موجود هستند. همچنین کدهای آورده شده در این گزارش نیز در کنار گزارش قرار دارند. از صبر و بردباری شما در مطالعه این مستند صمیمانه سپاسگزاریم.

با احترام

سروش جهان‌زاد، علی حاتمی تاجیک — بهار ۱۴۰۰

۲ طراحی UART

طراحی شده از چهار بخش اصلی تشکیل شده است UART

- تولید کننده سیگنال enable با توجه به بیت ریت
- فرستنده
- گیرنده
- کنترل کننده سه بخش قبل

کارکرد کلی دستگاه هم بدین صورت است که ابتدا کاربر عدد ورودی خود را اعلام می کند و دکمه شروع به کار دستگاه را می زند. سپس فرستنده از پایه TX خود ابتدا یک سیگنال صفر به معنای شروع می فرستد. سپس یک بیت parity مربوط به داده ورودی را ارسال می کند و سپس به ترتیب از بیت پر ارزش به بیت کم ارزش را ارسال می کند. در آن سمت گیرنده وقتی بیت صفر ابتدایی را می گیرد کار خود را شروع می کند. این بخش هم یک سیگنال فعالیت به اندازه بیت ریت دریافت می کنه و هنگامی که روی کلاک متناسب با ریت می رسد بیت های بعدی را می خواند. این کار که اولین بیتی هم که خوانده می شود روی بیت ریت باشد به این مسئله که بیت ها تقریباً در وسط ارسال خوانده شوند و روی لبه ها نباشد کمک خواهد کرد (یعنی احتمال اینکه اولین صفر روی لبه خوانده شود بسیار کم است). در نهایت که کار ماژول تمام شد پری تی آن چک می شود که مشخص شود در فرایند ارسال و دریافت اشکالی پیش آمده است یا نه. در ادامه کار هر قسمت به صورت جداگانه و کامل توضیح داده شده است.

لازم به ذکر است که نرخ کلاک ورودی به این ماژول باید ۱۰۰ مگاهرتز باشد!

Transmitter

این ماژول یک سیگنال کلاک، یک سیگنال فعال، یک سیگنال شروع، یک سیگنال اکتیو لو ریست و یک عدد ۷ بیتی ورودی می‌گیرد. خروجی‌های آن هم یک سیگنال busy که نشان‌دهنده این است که ماژول مشغول ارسال داده است و یک سیگنال tx که همان داده خروجی است خروجی دارد.

در لبه پایین رونده ریست کارهای مربوط به ریست انجام می‌شود. اگر استارت فشرده شده باشد، وضعیت را از ریلکس به ارسال بیت استارت تغییر می‌دهد. اگر در وضعیت ارسال بیت استارت باشد یک صفر می‌فرستد و به حالت بعد می‌رود. در حالت بعد دیتای از قبل ساخته شده را از رجیستر می‌خواند هشت بار و سپس به قسمت بعد می‌رود. در نهایت یک بیت استاپ می‌فرستد، مدار را به حالت اولیه در می‌آورد و منتظر سیگنال استارت بعدی می‌ماند.

```
module transmitter (tx, clk, tx_en, [7:0] data_in, start, busy, resetN);
    output reg tx, busy;
    input wire start, resetN, clk, tx_en, ;
    input [6:0] data_in;
    reg [7:0] data_send;
    reg [1:0] current_state = 0;
    reg [2:0] pos = 0;
    reg isStarted = 0;
    assign busy = ~(current_state == 2b'00); // if we are not relaxing, we are busy =)
    always @ (posedge clk or negedge resetN or posedge start) begin
        if (~resetN) begin // if module was reset
            current_state <= 2b'00; // go to Relax State
            isStarted <= 1b'0; // make not started
            tx <= 1b'1; // make tx 1
        end else if (start && ~isStarted) begin
            isStarted <= 1; // set started flag
            current_state <= 2b'01; // Current State to Send Start Bit
            data_send <= {~^data_in,data_in}; // set data to send
        end else if (current_state == 2b'01) begin // If in Send Start Bit State
            if (tx_en) begin // If we are on rate
                tx <= 1b'0; // send start bit
                current_state <= 2b'10; // go to next state - send data
            end
        end else if (current_state == 2b'10) begin // if in send data state
            if (tx_en) begin // if we are on rate
                tx <= data_send[7-pos]; // make tx the data in send position
                pos <= pos + 1; // add to position
                if (pos == 3b'111)begin // if we reach position 7
                    current_state <= 2b'11 // go to next state - send stop bit
                end
            end
        end else if (current_state == 2b'11) begin // if in send stop bit state
            if (tx_en) begin // if we are on rate
                tx <= 1b'1; // send stop bit (1)
                current_state <= 2b'00; // go to relax state
            end
        end
    end
end
endmodule
```

Receiver

این ماژول دریافت پیام را انجام می‌دهد. ورودی‌های این ماژول شامل ورودی سریال rx، کلاک دریافت‌کننده، بیت rx_en (برای enable) و بیت resetN (برای ریست آنسکرون اکتیولو) هستند. در هنگام دریافت پیام، سیگنال busy روشن و در غیر آن، خاموش است. پس از دریافت پیام، ۷ بیت داده‌ی خروجی در ۷ بیت سمت راست خروجی data_out قرار می‌گیرند و سیگنال parity نشانگر بیت parity دریافت شده از مبدا است. سیگنال خروجی error نیز هنگامی فعال می‌شود که زوجیت داده‌ی دریافت‌شده با بیت parity دریافت شده متفاوت باشد.

این ماژول دو حالت دارد که با سیگنال state نگهداری می‌شوند. در حالتی که این سیگنال برابر صفر باشد، ماژول در انتظار دریافت پیام است و تا زمانی که ورودی rx صفر شود، همین‌طور باقی می‌ماند. در هنگامی که ورودی rx_en روشن باشد و سیگنال شروع داده شود (بیت rx در یک کلاک صفر شود)، وارد حالت دوم (state = 1) می‌شویم. در این حالت در هر کلاک (به شرط روشن بودن rx_en) یک بیت از ورودی rx می‌خوانیم و از راست وارد رجیستر data می‌کنیم (همه‌ی بیت‌های این رجیستر یک واحد به سمت چپ شیف‌ت داده می‌شوند). این کار را برای ۹ سیکل ساعت انجام می‌دهیم. پس از آن، بیت شماره ۷ در این رجیستر (هشتم از سمت راست و اول از سمت چپ، با توجه به ۸ بیتی بودن رجیستر data) برابر سیگنال parity دریافت شده و بیت‌های شماره صفر تا شش همان ۷ بیت داده‌ی دریافت‌شده خواهند بود. دقت می‌کنیم که خروجی parity به بیت شماره ۷ و خروجی data_out به بیت‌های صفر تا شش در رجیستر data وصل هستند. پس از این عملیات، ماژول هنگام دریافت بیت دهم (بیت پایان) دوباره به حالت انتظار برای دریافت برمی‌گردد و آماده‌ی دریافت یک پیام جدید خواهد بود.

```
module receiver(rx, clk, data_out, rx_en, resetN, error, busy, parity);
    input rx, clk, rx_en, resetN;
    output [7:0] data_out;
    output error, busy, parity;
    reg state; // 0 is ready, 1 is receiving
    reg [3:0] counter;
    reg [7:0] data;
    assign busy = state;
    assign error = (data[7] == ^data[6:0]);
    assign data_out = {1'b0, data};
    assign parity = data[7];
    always @(posedge clk or negedge resetN) begin
        if (~resetN) begin
            state <= 0;
            data <= 0;
        end
        else if (rx_en) begin
            if (state == 0) begin // start
                if (~rx) begin
                    state = 1;
                end
                counter <= 0;
            end
            else begin // state == 1
                if(counter < 8) begin // 0 through 7 -> start, parity, data[6:0]
                    data <= {data[6:0], rx};
                end
                else begin // stop
                    state <= 0;
                end
                counter <= counter + 1;
            end
        end
    end
endmodule
```

Bit Rate Converter

این ماژول مسئول ساخت پالس فعال برای دریافت کننده و فرستنده است. به این صورت عمل می‌کند که یک پارامتر اندازه کلاک بر بیت ریت دارد و یک شمارنده دارد که با هر کلاک آنرا اضافه می‌کند و هر وقت این شمارنده به این عدد رسید یک پالس یک کلاک فعال تولید میکند و این روند به همین شکل ادامه پیدا خواهد کرد.

```
module bitrate_converter(  
    clk,  
    rx_en,  
    tx_en,  
    resetN  
);  
parameter baudrate = 100000000 / 9600; // 100 MHz clock, baud : 9600  
input clk, resetN;  
output wire rx_en, tx_en;  
reg[15:0] buadCountRX, buadCountTX;  
reg[15:0] rate = baudrate;  
// RX RATE  
always @ (posedge clk or negedge resetN) begin  
    if (~resetN) begin  
        buadCountRX <= 16'b1;  
    end else if (rx_en) begin  
        buadCountRX <= 16'b1;  
    end else begin  
        buadCountRX <= buadCountRX + 1'b1;  
    end  
end  
assign rx_en = (buadCountRX == rate);  
// TX RATE  
always @ (posedge clk or negedge resetN) begin  
    if (~resetN) begin  
        buadCountTX <= 16'b1;  
    end else if (tx_en) begin  
        buadCountTX <= 16'b1;  
    end else begin  
        buadCountTX <= buadCountTX + 1'b1;  
    end  
end  
assign tx_en = (buadCountTX == rate);  
endmodule
```

UART

این ماژول مسئول هماهنگی بین فرستنده و گیرنده است. یک نقش واسطه بین اجزاء را بازی می‌کند. این ماژول عدد ورودی، کلاک، سیگنال شروع، سیگنال ریست و RX را به عنوان ورودی دریافت می‌کند و در خروجی سیگنال‌های درگیر بودن هر کدام از خط‌های فرستنده و گیرنده، TX، خطای مربوط به پیریتی بیت، خود پیریتی بیت و دیتای خروجی را به عنوان خروجی دارد.

```
module uart (data_in, data_out,
    tx,rx,
    resetN,clk,
    error,parity_out, rx_busy, tx_busy,start);
    input [7:0]data_in;
    output [7:0] data_out;
    input resetN, clk;
    input rx, start;
    output tx, error, parity_out, tx_busy,rx_busy;
    wire rx_enable, tx_enable;

    bitrate_converter bitrate_converter_inst(
        .clk(clk),
        .rx_en(rx_enable),
        .tx_en(tx_enable),
        .resetN(resetN)
    );

    receiver receiver_inst(
        .rx(rx),
        .clk(clk),
        .data_out(data_out),
        .rx_en(rx_enable),
        .resetN(resetN),
        .error(error),
        .busy(rx_busy),
        .parity(parity_out)
    );

    transmitter transmitter(
        .tx(tx),
        .clk(clk),
        .tx_en(tx_enable),
        .data_in(data_in),
        .start(start),
        .busy(tx_busy),
        .resetN(resetN)
    );
endmodule
```

۳ تست مدار

برای تست این مدار دو عدد از دستگاه‌های طراحی شده را به یکدیگر متصل می‌کنیم. دو عدد ورودی به هر کدام می‌دهیم و سیگنالهای استارت آنها را می‌زنیم اما به این صورت که یکی را در حین انجام کار دیگری می‌زنیم. دو سیم هم نیاز داریم تا پایه‌های فرستنده و گیرنده یکدیگر را به هم وصل کنیم. در ادامه کد مورد تست و نتیجه تست آمده است:

```
module uart_tb();
    reg clk = 0;
    reg [7:0] data_in1 = 164;
    reg start1 = 0, resetN1 = 0;
    wire [7:0] data_out1;
    wire error1, parity_out1, tx_busy1, rx_busy1;
    wire wire1, wire2;
    uart u1(data_in1, data_out1, wire1, wire2, resetN1, clk,
    error1, parity_out1, rx_busy1, tx_busy1, start1);
    reg [7:0] data_in2 = 354;
    reg start2 = 0, resetN2 = 0;
    wire [7:0] data_out2;
    wire error2, parity_out2, tx_busy2, rx_busy2;
    uart u2(data_in2, data_out2, wire2, wire1, resetN2, clk,
    error2, parity_out2, rx_busy2, tx_busy2, start2);
    initial begin
        forever begin
            #5 clk = ~clk;
        end
    end
    initial begin
        #10
        resetN1 = 1;
        resetN2 = 1;
        start1 = 1;
        #40 start1 = 0;
        #5 start2 = 1;
        #5 start2 = 0;
        #1000000000 $finish;
    end
end
endmodule
```

